Titre: Title:	Étude et séparation des inégalités valides pour des problèmes de partitionnement et de couverture
Auteur: Author:	Mounira Groiez
Date:	2013
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Groiez, M. (2013). Étude et séparation des inégalités valides pour des problèmes de partitionnement et de couverture [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1170/

UNIVERSITÉ DE MONTRÉAL

ÉTUDE ET SÉPARATION DES INÉGALITÉS VALIDES POUR DES PROBLÈMES DE PARTITIONNEMENT ET DE COUVERTURE

MOUNIRA GROIEZ
DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES DE L'INGÉNIEUR)
JUIN 2013

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

ÉTUDE ET SÉPARATION DES INÉGALITÉS VALIDES POUR DES PROBLÈMES DE PARTITIONNEMENT ET DE COUVERTURE

présentée par : GROIEZ Mounira
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. EL HALLAOUI Issmail, Ph.D., président
M. DESAULNIERS Guy, Ph.D., membre et directeur de recherche
Mme MARCOTTE Odile, Ph.D., membre et codirectrice
M. LAPORTE Gilbert, Ph.D., membre
Mme LABBÉ Martine, Ph.D., membre

À mes parents.
À la mémoire de mes frères Mohamed-Said et Lahcène.

REMERCIEMENTS

Mes vifs remerciements à M^{me} Odile Marcotte et M. Guy Desaulniers pour leur encadrement, leur bienveillance, leur disponibilité, leur confiance et leurs précieux conseils.

Je remercie M. Ahmed Hadjar, pour toute l'aide qu'il m'a apportée dans la première partie de mon travail.

Je remercie les membres du jury qui m'ont fait l'honneur d'accepter de juger mon travail.

Je remercie tout le personnel du GERAD, en particulier Pierre, Serge et Francine pour leur aide.

Je remercie les membres de ma famille et mes amis, en particulier Assia, Fayçal, Sarah, Amira, Alain, Houcine et Issmail pour leur patience, leur gentillesse et pour leurs encouragements.

Mes très chaleureux remerciements à mes parents, qui tout au long de ma vie, ont su m'encourager et me soutenir.

Qu'ils trouvent tous ici, le témoignage de ma reconnaissance et l'expression de ma profonde gratitude.

RÉSUMÉ

Cette thèse s'intéresse aux inégalités valides et leurs algorithmes de séparation pour les problèmes de partitionnement et de couverture suivants : le problème d'horaires de véhicules avec plusieurs dépôts (**M**ultiple **D**epot **V**ehicle **S**cheduling **P**roblem, MDVSP), le problème de partitionnement d'ensemble (**S**et **P**artitioning **P**roblem, SPP) et le problème du transversal de circuits de cardinalité minimale (**MIN**imum cardinality **F**eedback **V**ertex **S**et problem, MIFVS). Notons que les inégalités valides introduites sont aussi applicables à d'autres problèmes ayant des structures similaires aux problèmes étudiés.

Dans la première partie de cette thèse, nous présentons un algorithme combinant la méthode d'élimination de variables, la méthode de séparation et évaluation progressive et la méthode de génération de plans coupants pour la résolution du MDVSP. Nous utilisons la formulation mathématique de multi-flot dans les réseaux, proposée par Ribeiro et Soumis (1994). Pour réduire la taille du problème nous débutons par l'application de la méthode d'élimination de variables. Dans la première phase, nous appliquons une proposition énoncée par Nemhauser et Wosley (1988) et dans la deuxième phase, nous utilisons le principe introduit par Irnich *et al.* (2010) mais sans génération de colonnes. Ensuite, afin d'améliorer la valeur de la borne inférieure, nous décrivons une méthode de génération de plans coupants. Cette dernière est basée sur une généralisation des inégalités valides introduites par Hadjar *et al.* (2006). Pour séparer ces inégalités valides, nous proposons d'identifier deux structures particulières propres au MDVSP. Par la suite, nous comparons la performance de différentes stratégies de notre algorithme à l'aide des instances générées aléatoirement. Finalement, nous concluons qu'en général notre algorithme est meilleur que le solveur commercial CPLEX.

La première méthode de séparation proposée pour identifier la première structure, que nous nommons structure de cycle impair conflictuel, utilise la similitude entre le problème du stable et le MDVSP. Notre méthode consiste à construire un graphe de conflit à l'aide d'un sous-ensemble d'arcs possédant certaines propriétés dans le multigraphe représentant le MDVSP. Nous prouvons qu'un trou impair dans le graphe de conflit correspond à un cycle impair conflictuel dans le multigraphe du MDVSP. Afin d'identifier les trous impairs nous adaptons l'algorithme proposé par Nemhauser et Sigismondi (1992). En outre, nous démontrons que chaque cycle impair conflictuel permet d'obtenir une inégalité valide violée par la solution fractionnaire courante. Afin d'enrichir ces inégalités, nous décrivons une procédure de liftage.

La deuxième méthode de séparation proposée pour identifier la deuxième structure, que nous nommons structure de coupe impaire, exploite la ressemblance entre le problème de

couplage et le MDVSP et, par conséquent, entre les inégalités valides du MDVSP et les inégalités d'Edmonds (1965). La relation entre les deux problèmes est basée sur le principe de conflit, pour le problème de couplage, deux arêtes adjacentes sont toujours en conflit et pour le MDVSP deux sous-chemins adjacents peuvent être en conflit (s'ils ont la même source ou le même puits ou s'ils sont de couleurs différentes), donc si nous remplaçons un sous-chemin par une arête, alors les inégalités d'Edmonds engendrent des inégalités valides du MDVSP. Afin d'identifier des structures de coupes impaires, nous construisons un graphe de chemins réguliers, où un chemin régulier est une suite d'arcs de même couleur. Ensuite, nous formulons le problème de séparation comme un programme linéaire en nombres entiers et nous prouvons que certaines solutions entières de ce dernier nous permettent d'identifier des inégalités de coupe impaire. Afin de l'enrichir, nous proposons une procédure de liftage. Nous démontrons que chaque coupe impaire liftée correspond à une inégalité valide pour le MDVSP.

Dans la deuxième partie de cette thèse, nous proposons la résolution du problème de partitionnement d'ensemble par une méthode combinant la méthode de séparation et évaluation progressive et la méthode de génération de plans coupants. Notons que nous nous intéressons au problème de partitionnement pur. Nous introduisons deux nouvelles classes d'inégalités valides : nommées les inégalités de type I et de type II, pour le problème de partitionnement. En outre, nous formulons le problème de séparation de chaque classe d'inégalités valides comme un programme linéaire en nombres entiers et nous montrons que certaines solutions entières de ce dernier permettent d'identifier des inégalités valides pour le problème de partitionnement. Ensuite, nous proposons de séparer les inégalités de clique en résolvant un autre problème auxiliaire en nombres entiers. Enfin, nous analysons l'efficacité de notre algorithme en utilisant quelques instances de Hoffman et Padberg (1993) et d'autres générées aléatoirement à l'aide du générateur proposé par Lewis *et al.* (2008). Finalement, nous concluons que notre algorithme est plus efficace que CPLEX lorsque la densité du problème à résoudre est supérieure à 16% et ne l'est pas dans le cas contraire.

Dans la dernière partie de cette thèse, nous présentons une méthode combinant la génération de contraintes et des plans coupants afin de résoudre une variante du problème de transversal de circuits de cardinalité minimale. Ce dernier est relié à l'analyse des dictionnaires des langues naturelles. Nous nous intéressons à un des problèmes de linguistique qui consiste à déterminer le nombre minimum de mots à connaître pour comprendre toutes les définitions d'un dictionnaire donné. Ce problème est formulé comme un problème de transversal de circuits de cardinalité minimale. Nous décrivons des méthodes de séparation de trois classes d'inégalités valides : deux classes particulières des inégalités de Chvátal-Gomory et les inégalités de clique de cardinalité 3. Nous formulons le problème de séparation de chaque

classe comme un programme linéaire en nombres entiers et nous démontrons que certaines solutions entières permettent d'identifier des inégalités valides pour le problème à résoudre. Enfin, nous présentons et nous analysons les résultats des tests effectués à l'aide des exemplaires extraits de dictionnaires de langue anglaise. Malheureusement, les résultats obtenus ne sont pas concluants étant donné que pour certaines instances nous n'avons pas réussi à trouver les solutions optimales en utilisant le solveur commercial, ni à prouver l'optimalité en utilisant notre algorithme. Néanmoins, nous constatons que l'ajout des inégalités valides permet de trouver de meilleures solutions réalisables dans le temps limite imparti et dans certains cas à trouver la solution optimale.

Les travaux présentés dans cette thèse constituent une démarche basée sur les caractéristiques du problème à résoudre. Notre objectif est de présenter de nouveaux algorithmes efficaces pour la séparation des inégalités valides, permettant l'amélioration de la valeur de la relaxation linéaire, mais « bien qu'on ait du cœur à l'ouvrage, l'art est long et le temps est court ». (Charles Baudelaire, Les fleurs du mal).

ABSTRACT

In this thesis, we study valid inequalities and we propose branch-and-cut algorithms for some classical problems such as : the Multiple Depot Vehicle Scheduling Problem (MDVSP), the Set Partitioning Problem (SPP) and the MINimum cardinality Feedback Set problem (MINFVS).

In the first part of this thesis, we present an algorithm combining branch-and-bound, cutting planes and variable fixing to solve the MDVSP. We use the multicommodity flow model of the MDVSP, proposed by Reibero and Soumis (1994). In the first step, we apply a proposition to fix the variables that was introduced by Nemhauser and Wosley (1988) and used by Hadjar *et al.* (2006). In order to eliminate more variables, we use the same rule as Irnich *et al.* (2010) but without column generation. In the second step, we describe a method for separating the valid inequalities introduced by Hadjar *et al.* (2006) and generalized in the present work. Our method identifies two special structures corresponding to valid inequalities. Then we present the results of computational experiments comparing the performance of several strategies using the random instances. We conclude that our algorithm is better than the commercial software.

The separation procedure that we propose to identify the first structure, called odd conflictual cycle, uses the similarity between the MDVSP and the stable set problem. We start by creating a conflict graph of the multigraph representing the MDVSP, we consider only the arcs in conflict. Then we prove that each odd hole in this conflict graph is an odd conflictual cycle in the multigraph representing the MDVSP. In order to identify odd holes we adopt the polynomial algorithm proposed by Nemhauser and Sigismondi (1992). Moreover we prove that every odd conflictual cycle identifies a valid inequality violated by the fractional solution. We also present a lifting procedure in order to enrich the valid inequalities.

The separation procedure that we present to identify the second structure, called odd set, uses the relationship between the MDVSP and the matching problem, consequently the relationship between the valid inequalities for the MDVSP and the blossom inequalities introduced by Edmonds (1965). In the matching problem two adjacent edges are always in conflict and for the MDVSP two adjacent subpaths may be in conflict (if they are the same source or the same sink or of different colours). In order to identify the odd set, we use the regular path graph, where a regular path is a sequence of the arcs with the same colour. Then we formulate the separation problem as an integer program and we prove that each integer solution identifies, under certain conditions, an odd set. In addition, we present a lifting procedure and we prove that each inequality obtained by lifted odd set is a valid inequality

for the MDVSP.

In the second part of this thesis, we propose an algorithm combining branch-and-bound and the cutting planes method to solve the set partitioning problem. Note that we are interested in the pure partitioning problem. We derive new classes of valid inequalities for the problem. Moreover, we present a separation procedure for each class by solving an auxiliary integer program using a fractional solution of the set partitioning problem. Then we prove that each integer solution of this program, under certain conditions, is a valid inequality for the problem. We lift this inequality with variables of value 0. Also we propose to separate a clique inequality by using another auxiliary integer program and we prove that each integer solution of this program, under certain conditions, is a clique inequality for the set partitioning problem. Finally, we present the results of a computational study and we compare our results with the commercial software. Note that we use instances of Hoffman and Padberg (1993) and other randomly generated instances using the generator proposed by Lewis *et al.* (2008). We conclude that our algorithm is better than the commercial software when the density of the problem is greater than 16% and is not otherwise.

In the last part of this thesis, we present a method combining constraint generation and cutting planes generation in order to solve a class of feedback vertex set problem arising in the analysis of the natural language dictionaries. A dictionary is a set of definitions, where a definition consist of one or several sentences represented by the sequence of words. We are interested in the linguistic problem of determining the minimum number of words to know in order to understand all definitions in the dictionary. We propose to separate two classes of Chvátal-Gomory inequalities and clique inequalities of cardinality 3 by solving an auxiliary integer program for each of them. Also we prove that each integer solution of the auxiliary program identifies a valid inequality for the feedback vertex set problem. Finally we present and analyze the resultats of computational experiments using the dictionaries of the English language are available on the Word Wide Web. Unfortunately, the results are not conclusive because we are not able to find optimal solution using a solver commercial or prove optimality using our algorithm. However, we observe that adding the valid inequalities permits to find better feasibles solutions within the time limit prescribed.

The work presented in this thesis is an approach based on the characteristics of the problem. Our objective is to present new efficient algorithms for the separation of valid inequalities in order to improve the value of the linear relaxation.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1

INTRODUCTION

Un problème d'optimisation combinatoire consiste à déterminer la meilleure solution parmi toutes les solutions admissibles. L'ensemble de toutes les solutions admissibles est nommé l'ensemble des solutions réalisables. En général, cet ensemble est fini mais contient un très grand nombre d'éléments, il est même impossible d'énumérer toutes les solutions réalisables des problèmes de grande taille. Il existe deux familles de méthodes de résolution des problèmes d'optimisation : les méthodes exactes et les méthodes heuristiques. Les méthodes exactes, contrairement aux méthodes heuristiques, assurent l'optimalité de la solution trouvée. Néanmoins, le temps de résolution est généralement exponentiel en fonction de la taille du problème traité.

En pratique, les méthodes exactes utilisent une ou plusieurs combinaisons des stratégies suivantes : la méthode d'élimination de variables, la méthode de séparation et évaluation progressive, la génération de colonnes et la génération de plans coupants. La méthode d'élimination de variables consiste à fixer une variable à l'une de ces bornes si elle vérifie un critère donné, cette méthode permet de réduire la taille du problème à résoudre. La méthode de séparation et évaluation progressive (ou « branch-and-bound ») est basée sur le calcul de bornes inférieures et supérieures pour éviter d'explorer un intervalle ne contenant pas de meilleures solutions que celles déjà identifiées. La méthode de génération de colonnes combinée à la méthode de séparation et évaluation progressive (ou « branch-and-price ») est appliquée pour résoudre les problèmes qui possèdent un très grand nombre de variables. Au niveau de chaque noeud de l'arbre de branchement, la méthode de génération de colonnes fait appel à un problème maître qui ne contient qu'un sous-ensemble de variables et un sous-problème qui génère les variables qu'il faut ajouter au problème maître à chaque étape. La méthode de génération de plans coupants combinée à la méthode de séparation et évaluation progressive (ou « branch-and-cut ») consiste à réduire le domaine réalisable de la relaxation linéaire en ajoutant de nouvelles contraintes valides pour le problème en nombres entiers mais violées par la solution fractionnaire courante. Ces contraintes sont nommées des inégalités valides ou plans coupants et la méthode utilisée pour les déterminer est nommée méthode de séparation des inégalités valides. Un problème de séparation consiste à déterminer si la solution courante appartient à l'enveloppe convexe des solutions entières réalisables, sinon à générer des inégalités séparant cette solution de l'enveloppe convexe. Il existe rarement des algorithmes polynomiaux pour la résolution du problème de séparation des inégalités valides

associé à un problème NP-complet.

Au cours des dernières années, plusieurs problèmes d'optimisation combinatoire ont été résolus à l'aide de méthodes de type « branch-and-cut ». L'expérience a montré que les inégalités valides sont plus efficaces si le problème de séparation exploite la particularité de la structure du problème étudié. Généralement, les problèmes pratiques sont modélisés comme des programmes linéaires en nombres entiers qui combinent des contraintes définissant des structures simples et d'autres contraintes sans structure particulière. L'idée est d'utiliser les structures simples afin de définir des classes d'inégalités valides propres au problème à résoudre.

Dans le présent travail, nous étudions les problèmes de séparation de trois problèmes de partitionnement et de couverture, à savoir le problème d'horaires de véhicules avec plusieurs dépôts (**M**ultiple **D**epot **V**ehicle **S**cheduling **P**roblem, MDVSP), le problème de partitionnement d'ensemble (**S**et **P**artitioning **P**roblem, SPP) et le problème du transversal de circuits de cardinalité minimale (**MIN**imum cardinality **F**eedback **V**ertex **S**et problem, MINFVS). Les trois problèmes se formulent comme des programmes linéaires en nombres entiers, bien que leurs formulations mathématiques soient simples, ils sont NP-complets, c'est-à-dire qu'il est peu probable qu'il existe d'algorithmes polynomiaux pour les résoudre.

Dans le chapitre 2, nous définissons chaque problème étudié, présentons une formulation mathématique pour chacun et survolons la littérature pertinente. Nous présentons les plus importantes inégalités valides introduites pour chacun des problèmes. Nous décrivons aussi les méthodes directe et bidirectionnelle d'élimination de variables existantes dans la littérature.

Dans le chapitre 3, nous généralisons la notion de sous-multigraphe épineux introduite par Hadjar *et al.* (2006) pour le MDVSP. Nous présentons deux structures particulières permettant l'identification d'inégalités valides propres au MDVSP. Nous décrivons un algorithme de complexité polynomiale pour la séparation du premier type d'inégalités et nous formulons un problème auxiliaire en nombres entiers pour la séparation du deuxième. Nous décrivons aussi un algorithme glouton pour la séparation des inégalités de clique de cardinalité 3. Nous proposons un algorithme de type « branch-and-cut » pour la résolution du MDVSP. Nous utilisons des instances générées aléatoirement pour tester les différentes versions de notre algorithme. Les résultats des tests nous permettent de tirer des conclusions intéressantes concernant l'éfficacité de notre algorithme.

Dans le chapitre 4, nous proposons deux nouvelles classes d'inégalités valides propres au SPP. Nous démontrons qu'il existe une relation entre les deux classes. Le problème de séparation de chaque classe d'inégalités valides est formulé comme un problème auxiliaire en nombres entiers. Nous suggérons de séparer les inégalités de clique en résolvant un programme linéaire en nombres entiers. Nous élaborons un algorithme de type « branch-and-cut » pour

la résolution du SPP. Pour tester notre algorithme, nous utilisons des instances de Hoffman et Padberg (1993) et des instances aléatoires créées par le générateur de Lewis *et al.* (2008). Nous terminons par quelques conclusions.

Dans le chapitre 5, nous étudions un problème de linguistique qui consiste à déterminer le nombre minimum de mots à connaître pour comprendre tous les mots d'un dictionnaire. Ce problème est formulé comme un problème du transversal de circuits de cardinalité minimale. Nous présentons la description et la formulation du problème. Nous adaptons trois classes d'inégalités valides existantes dans la littérature : les inégalités d'ensemble impair, un cas particulier des inégalités de Chvátal-Gomory de rang 1 et les inégalités de clique. Le problème de séparation de chaque classe d'inégalités valides est formulé comme un programme linéaire en nombres entiers. Nous essayons de résoudre le problème du transversal de circuits de cardinalité minimale par un algorithme de type « branch-and-cut ». Nous testons notre algorithme en utilisant des dictionnaires de la langue anglaise et nous terminons par une conclusion.

Dans le chapitre 6, nous présentons une synthèse des travaux présentés dans cette thèse, les difficultés rencontrées et des idées pour améliorer les résultats obtenus dans le présent travail.

CHAPITRE 2

REVUE DE LITTÉRATURE

2.1 Le problème d'horaires de véhicules avec plusieurs dépôts (MDVSP)

Considérons un ensemble de n trajets (ou tâches) $\{T_1, T_2,...,T_n\}$, où chaque trajet T_i débute au temps a_i et se termine au temps b_i. Les véhicules effectuant les trajets sont logés dans des dépôts. Soit K l'ensemble des indices des dépôts considérés, où $K = \{1, 2, ..., |K|\}$. Nous supposons que le dépôt D_k a une capacité de v_k véhicules. Soit t_{ij} le temps nécessaire à un véhicule pour se rendre de l'emplacement où le trajet T_i se termine à l'emplacement où le trajet T_j doit commencer. La paire de trajets consécutifs (T_i, T_j) est dite compatible (ou réalisable) si un véhicule peut effectuer T_j immédiatement après T_i, c'est-à-dire que la relation $b_i + t_{ij} \leq a_j$ est vérifiée. Une tournée (ou un horaire) est une suite de trajets compatibles effectués par le même véhicule, qui doit débuter et retourner au même dépôt.

Le coût associé à chaque paire de trajets compatibles (T_i, T_j) est $c_{ij} \geq 0$; notons que $c_{ij} = +\infty$ si (T_i, T_j) est non réalisable. Pour chaque trajet T_i et chaque dépôt D_k, soit $c_{n+k,i}$ (resp. $c_{i,n+k}$) le coût non négatif engendré par l'utilisation d'un véhicule du dépôt D_k débutant (resp. terminant) sa tournée par le trajet T_i. Par conséquent, le coût total d'une tournée $(T_{i_1}, T_{i_2},..., T_{i_h})$ associé à un véhicule du dépot D_k est $c_{n+k,i_1} + c_{i_1,i_2} + \ldots + c_{i_h,n+k}$.

Le MDVSP consiste à trouver des tournées de véhicules de coût minimal possédant les propriétés suivantes :

- chaque trajet T_i, $i = 1, ..., n$, est effectué par exactement un véhicule ;
- chaque véhicule utilisé dans la solution couvre une suite de trajets (une tournée) dans laquelle toute paire de trajets consécutifs est compatible ;
- chaque véhicule, après avoir effectué ses tâches, revient au dépôt de départ ;
- le nombre de véhicules utilisés du dépôt D_k, $k \in K$, ne peut dépasser v_k ;
- le coût total de l'affectation est égal à la somme des coûts des tournées effectués par les véhicules.

La formulation que nous avons adoptée est celle utilisée par Ribeiro et Soumis (1994). Considérons un multigraphe $G = (V, A)$ orienté et acyclique, tel que V représente l'ensemble des sommets et A représente l'ensemble des arcs. L'ensemble des sommets V est la réunion de l'ensemble des tâches $\{T_1, T_2, ..., T_n\}$ et de celui des dépôts $\{D_1, D_2, ..., D_{|K|}\}$; l'ensemble des arcs, A, contient tous les arcs de la forme (D_k, T_i) ou (T_i, D_k) pour tout $k \in K$ et $i = 1, 2, \ldots, n$ et $|K|$ copies des arcs de la forme (T_i, T_j). Nous utilisons le triplet (i, j, k)

pour faire référence à un arc de G indiquant qu'un véhicule du dépôt k effectue la tâche T_j immédiatement après la tâche T_i. Notons que si $i = n + k$ et $j \leq n$ (resp. $i \leq n$ et $j = n + k$), alors (i, j, k) représente l'arc (D_k, T_j) (resp. (T_i, D_k)). À chaque arc $(i, j, k) \in A$, nous associons une variable binaire X_{ij}^k qui prend la valeur 1 si l'arc (i, j, k) appartient à un des chemins dans une affectation réalisable.

Dans ce qui suit, nous utiliserons la notation suivante :

- $\delta^+(i) = \{j \mid (i, j, k) \in A\}$
- $\delta^-(i) = \{j \mid (j, i, k) \in A\}$
- $\delta(i) = \delta^+(i) \cup \delta^-(i)$.

Le problème se formule alors comme un programme linéaire en nombres entiers.

$$
(MDVSP) \begin{cases}
\min \displaystyle\sum_{(i,j,k)\in A} c_{ij} X_{ij}^k & & (2.1) \\
\displaystyle\sum_{k\in K} \sum_{j\in\delta^+(i)} X_{ij}^k = 1 & i = 1, \ldots, n & (2.2) \\
\displaystyle\sum_{j=1}^{n} X_{n+k,j}^k \leq v_k & \forall k \in K & (2.3) \\
\displaystyle\sum_{j\in\delta^-(i)} X_{ji}^k - \sum_{j\in\delta^+(i)} X_{ij}^k = 0 & \forall k \in K,\ \forall i = 1, \ldots, n, n+k & (2.4) \\
X_{ij}^k \in \{0, 1\} & \forall (i, j, k) \in A & (2.5)
\end{cases}
$$

L'objectif (2.1) minimise le coût total d'une affectation réalisable. Les contraintes (2.2) assurent que chaque tâche soit effectuée par un seul véhicule. Les contraintes (2.3) garantissent que le nombre de véhicules quittant D_k est au plus égal à v_k. Les contraintes de conservation de flot en chaque tâche et en chaque dépôt sont représentées par les équations (2.4). Les contraintes (2.5) indiquent que les variables sont binaires.

Le MDVSP a fait l'objet de plusieurs travaux de recherche durant les dernières années. Plusieurs algorithmes ont été proposés utilisant la séparation et évaluation progressive, la génération de colonnes, la relaxation lagrangienne, etc. L'article de Desaulniers et Hickman (2007) présente une revue de littérature regroupant les plus récents et plus importants travaux publiés sur le MDVSP.

Il est aussi important de noter que certains chercheurs (la plupart d'entre eux) ont testé leurs algorithmes en utilisant des données générées aléatoirement. Par contre, d'autres chercheurs, dont Löbel (1998), Gintner *et al.* (2005) et Kliewer *et al.* (2006), ont utilisé des données qui sont adaptées à la réalité. Dans ces données, les réseaux étudiés sont moins complexes que ceux obtenus pour des données aléatoires.

Bertossi *et al.* (1987) ont montré que le MDVSP est NP-complet. Mais dans le cas où le

nombre de dépôts est égal à 1, le problème peut être résolu en temps polynomial. Carpaneto *et al.* (1989) ont décrit un algorithme de type séparation et évaluation progressive basé sur un critère de dominance (introduit par Fischetti et Toth (1988)) et une procédure additive permettant le calcul d'une borne inférieure additive.

Forbes *et al.* (1994) ont analysé une formulation comme programme linéaire à trois indices. Bianco *et al.* (1994) ont proposé une procédure additive qui calcule une borne inférieure en utilisant une solution heuristique du dual de la relaxation linéaire du MDVSP. Cette solution duale est utilisée pour réduire le nombre de variables du problème. Les auteurs décrivent ensuite un algorithme de type séparation et évaluation progressive pour résoudre le problème.

Ribeiro et Soumis (1994) ont présenté une nouvelle formulation mathématique du MDVSP, celle d'un modèle multi-flot dans les réseaux. Ils ont montré que la relaxation linéaire de cette formulation fournit une meilleure borne inférieure que celle obtenue par la procédure additive utilisée par Carpaneto *et al.* (1989). Les auteurs ont aussi reformulé le problème du MDVSP comme un problème de partitionnement d'ensemble et utilisé la technique de génération de colonnes pour le résoudre.

Löbel (1998) a proposé une approche heuristique basée sur la méthode de génération de colonnes et la relaxation lagrangienne pour la résolution du problème. Fischetti *et al.* (2001) ont présenté une étude polyédrale pour résoudre le problème du MDVSP. Les auteurs ont utilisé une seule variable pour chaque paire compatible et ont ajouté une autre classe de contraintes assurant que seulement les chemins réalisables font partie de la solution. Ensuite, ils ont ajouté une famille d'inégalités valides renforçant cette classe de contraintes.

Kliewer *et al.* (2006) ont introduit une nouvelle approche de modélisation en utilisant un réseau espace-temps. Le modèle proposé a permis de réduire la taille des problèmes réels étudiés. Gintner *et al.* (2005) ont utilisé la formulation proposée par Kliewer *et al.* (2006) et ont proposé une heuristique permettant de fixer des variables du problème. Les auteurs ont introduit la notion de groupe de dépôts, et ont montré que cette notion permet de réduire les temps de résolution pour des instances réelles.

Hadjar *et al.* (2006) ont proposé un algorithme de type « Branch-and-Price-and-Cut », combinant une procédure de type séparation et évaluation progressive, une procédure permettant l'élimination de variables, la génération de colonnes et la génération de plans coupants. Ils ont aussi montré que la solution de la relaxation linéaire du MDVSP contient beaucoup de cycles impairs. La stratégie d'élimination de variables utilisée par les auteurs a permis de fixer plus de 90% des variables pour la plupart des instances. En utilisant la notion de cycle impair, les auteurs ont défini des inégalités valides du MDVSP et ont proposé une procédure de liftage . Ils ont montré aussi que, sous certaines conditions, les inégalités valides liftées représentent des facettes du polytope du MDVSP.

Nous trouvons dans l'article de Pepin *et al.* (2009) un survol des plus importantes heuristiques utilisées pour résoudre le MDVSP. Les auteurs ont comparé les performances d'une heuristique de programmation en nombres entiers, d'une heuristique lagrangienne, d'une heuristique de génération de colonnes, d'une méthode de recherche à grands voisinages utilisant la génération de colonnes et d'une méthode de recherche tabou. Laurent et Hao (2009) ont proposé un algorithme de recherche locale itérée basée sur une procédure d'analyse du voisinage. Les auteurs ont montré que leur algorithme est plus performant que d'autres métaheuristiques utilisées pour résoudre le problème.

2.1.1 Le problème de couplage et les inégalités valides

Etant donné un graphe non orienté $G = (V, E)$, où V représente l'ensemble des sommets ($|V| = n$) et E représente l'ensemble des arêtes ($|E| = m$). Le problème de couplage consiste à déterminer un sous-ensemble d'arêtes deux à deux non adjacentes. Le problème est formulé comme un programme linéaire en nombres entiers.

$$(P) \begin{cases} \max & z = cx \\ s.c. & Ax \leq 1 \\ & x \in \{0,1\}^m. \end{cases}$$

où $c \in \mathbb{Z}^m$, A est la matrice d'incidence sommets-arêtes de dimension $n \times m$ associée au graphe $G = (V, E)$ et x_{ij}, $(i, j) \in E$, est une variable binaire qui prend la valeur 1 si l'arête (i, j) est choisie et 0 sinon.

Edmonds (1965) a montré que l'enveloppe convexe de l'ensemble des solutions entières du problème de couplage est décrite par les contraintes ci-dessus et par les contraintes de couplage (ou « blossom inequalities ») suivantes :

$$\sum_{(i,j)\in E | i,j\in W} x_{ij} \leq \left\lfloor \frac{|W|}{2} \right\rfloor \quad \forall\, W \subseteq V, \tag{2.6}$$

où $|W|$ est impair.

Padberg et Rao (1982) ont montré qu'une contrainte de couplage (2.6) est violée si et seulement si le graphe associé à la solution non entière trouvée contient une coupe impaire de capacité strictement inférieure à 1. Sinon la solution trouvée appartient à l'enveloppe convexe du problème (P) et il est impossible de la couper avec une inégalité (2.6). Les auteurs ont proposé une procédure de séparation de ces inégalités basée sur l'algorithme de Gomory et Hu (1961). Notons que le problème de la coupe impaire de capacité minimale consiste à trouver, dans un graphe $G = (V, E)$, un ensemble $U \subset V$ de cardinalité impaire tel que $\sum_{e\in\delta(U)} c_e$ est

minimale, où c_e est le poids de l'arête $e \in E$.

De la définition du problème de couplage, deux arêtes incidentes ne peuvent faire partie de la même solution réalisable. Par conséquent, ces deux arêtes sont en conflit. Hadjar *et al.* (2006) ont introduit la notion de conflit dans le MDVSP. Les auteurs ont exploité l'analogie entre le problème de couplage et le MDVSP pour proposer les inégalités de cycle impair pour le MDVSP.

2.1.2 Les inégalités de cycle impair (Hadjar *et al.* (2006))

Les inégalités valides introduites par Hadjar *et al.* (2006) pour le MDVSP sont nommées les inégalités de cycle impair. Dans cette section, nous présentons leur principe et nous commençons par définir quelques notions importantes introduites par les auteurs.

Définition 2.1.1 *Pour tout sous-ensemble A' de A, on définit*

- $Col(A') = \{\ k \in K \mid A'$ *contient au moins un arc de couleur* $k\ \}$,
- $\delta^-_{A'}(i)$ *(resp.* $\delta^+_{A'}(i)$*) = l'ensemble des arcs appartenant à* A' *tels que la destination de l'arc (resp. l'origine de l'arc) est le sommet* i *et*
- $\delta_{A'}(i) = \delta^-_{A'}(i) \cup \delta^+_{A'}(i)$.

Définition 2.1.2 *Soit (i, j, k) et (i', j', k') deux arcs adjacents dans le multigraphe G qui ne sont pas incidents aux dépôts (c'est-à-dire que $i, j, i', j' \leq n$). On dit qu'ils sont en conflit si $k \neq k'$ ou $i = i'$ ou $j = j'$.*

En d'autres termes, deux arcs adjacents sont en conflit s'ils ne peuvent pas appartenir à la même solution réalisable.

Définition 2.1.3 *On dit que le sous-multigraphe partiel $G' = (S, F)$ de G est conflictuel si :*

- $|S|$ *est impair,*
- S *ne contient aucun dépôt,*
- *tous les arcs adjacents dans F sont en conflit.*

Au niveau de chaque sommet $i \in S$, l'inégalité $\sum_{(i,j,k)\in\delta(i)} X^k_{ij} \leq 1$ est valide pour le MDVSP, puisque, selon la définition 2.1.3, tous les arcs adjacents dans F sont en conflit. Par conséquent, la somme des inégalités précédentes implique que $2 \sum_{(i,j,k)\in F} X^k_{ij} \leq |S|$ est vérifiée. Il s'ensuit que, étant donné un sous-multigraphe conflictuel $G' = (S, F)$, l'inégalité $\sum_{(i,j,k)\in F} X^k_{ij} \leq \left\lfloor \frac{|S|}{2} \right\rfloor$ est valide pour le MDVSP.

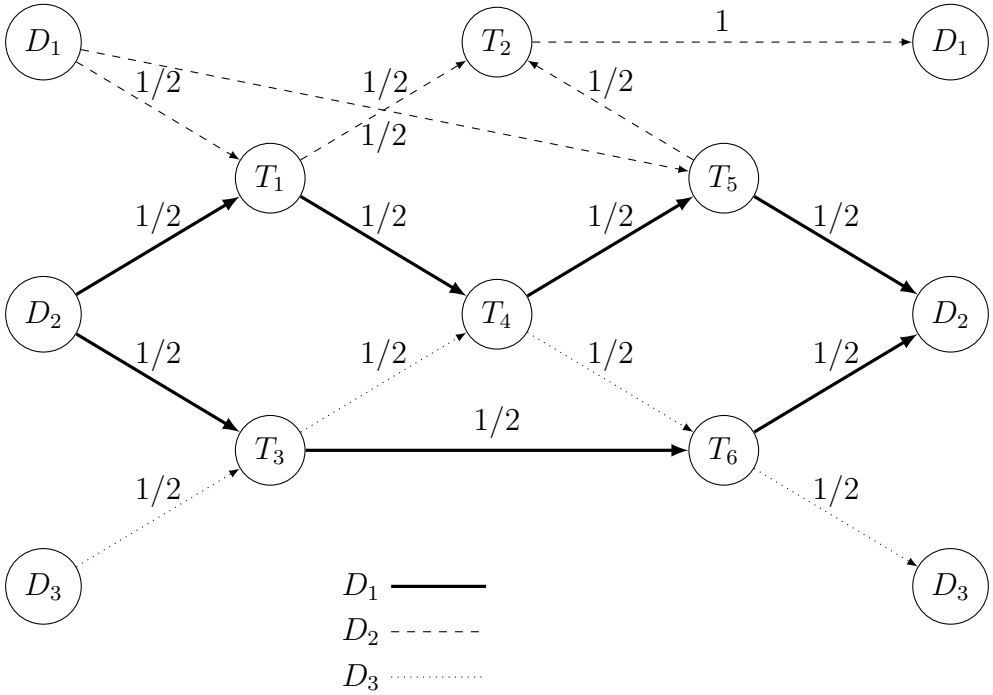

Figure 2.1 Exemple de solution fractionnaire de la relaxation linéaire du $MDVSP$

Malheureusement, toute solution fractionnaire ne contient pas un sous-multigraphe conflictuel. Par exemple, si nous considérons la solution fractionnaire de la figure 2.1, tirée de l'article de Hadjar *et al.* (2006), nous remarquons que (T_1, T_4) et (T_4, T_5) ne sont pas en conflit car T_4 est l'extrémité finale du premier arc et l'extrémité initiale du second arc et les arcs sont de même couleur. Par contre, (T_3, T_4) et (T_1, T_4) ainsi que (T_3, T_4) et (T_4, T_5) sont en conflit parce qu'ils ne sont pas de même couleur. Si nous considérons l'ensemble des sommets $S = \{T_1, T_2, T_3, T_4, T_5\}$ et l'ensemble des arcs $F = \{(T_1, T_2), (T_5, T_2), (T_4, T_5), (T_1, T_4), (T_3, T_4)\}$, alors l'inégalité $X^1_{12} + X^1_{52} + X^2_{14} + X^2_{45} + X^3_{34} \leq 2$ élimine la solution fractionnaire dans ce cas ; cette inégalité est donc valide pour le MDVSP. L'arc (T_3, T_4) qui crée le conflit au niveau du sommet T_4 est nommé épine et le sous-multigraphe induit par les sommets T_1, T_2, T_3, T_4, T_5 est appelé sous-multigraphe conflictuel épineux.

Définition 2.1.4 *Soit $G' = (S, F)$ un sous-multigraphe partiel de G. On dit que G' est un sous-multigraphe épineux si S peut être partitionné en S_1, S_2 et S_3 et F peut être partitionné en F_1 et F_2 de telle sorte que :*

- *$S_1 \cup S_2$ ne contient aucun dépôt,*
- *$|S_1|$ est impair et au moins égal à 3,*
- *$\delta^-_{F_2}(i) = \emptyset$ et $Col(\delta^-_{F_1}(i)) \bigcap Col(\delta^+_{F_1}(i)) = \emptyset$ pour tout $i \in S_1$,*

- $Col(\delta^-_{F_2}(i)) \bigcap Col(\delta^-_{F_1}(i)) = \emptyset$ *et* $Col(\delta^-_{F_2}(i)) \bigcap Col(\delta^+_{F_1}(i)) = \emptyset$ *pour tout* $i \in S_2$, *et*
- $\delta^-_F(i) = \emptyset$, $\delta^+_F(i) \subseteq F_2$ *et* $\delta^+_F(i) \neq \emptyset$ *pour tout* $i \in S_3$.

Proposition 2.1.5 *Soit* $G' = (S, F)$ *un sous-multigraphe épineux de* G, *où* $S = S_1 \cup S_2 \cup S_3$ *et* $F = F_1 \cup F_2$ *et les* S_i *et les* F_j *satisfont la définition précédente. Alors l'inégalité*

$$\sum_{(i,j,k)\in F} X^k_{i,j} \leq \left\lfloor \frac{|S_1|}{2} \right\rfloor + |S_2| \tag{2.7}$$

est valide pour le $(MDVSP)$.

Hadjar *et al.* (2006) ont démontré la proposition 2.1.5 en utilisant le principe du conflit entre les arcs au niveau des noeuds du sous-multigraphe épineux. Les auteurs ont utilisé une méthode d'énumeration pour la séparation des inégalités associées aux sous-multigraphes épineux.

2.2 Problème de partitionnement d'ensemble (SPP)

Considérons une matrice $A = (a_{ij})$ dont les coefficients sont égaux à 0 ou 1. Soit $I = \{1, \ldots, m\}$ l'ensemble des lignes de la matrice A et $J = \{1, \ldots, n\}$ l'ensemble des colonnes de la matrice A. Soit c_j, pour $j \in J$, le coût de la colonne $j \in J$. L'élément a_{ij} est égal à 1 si la colonne $j \in J$ couvre la ligne $i \in I$ et 0 sinon. Le problème de partitionnement d'ensemble consiste à déterminer un sous-ensemble de colonnes $N \subseteq J$, de coût minimal, tel que chaque ligne $i \in I$ est couverte exactement par une colonne $j \in N$. Nous associons à chaque colonne j une variable binaire x_j telle que x_j prend la valeur 1 si la colonne j est sélectionnée et 0 sinon. Le SPP est formulé comme un programme linéaire en nombres entiers.

$$(P) \begin{cases} \min \quad z = \sum_{j\in J} c_j x_j & & (2.8) \\ s.c. \quad \sum_{j\in J} a_{ij} x_j = 1 & \forall i \in I & (2.9) \\ x_j \in \{0,1\} & \forall j \in J & (2.10) \end{cases}$$

L'objectif (2.8) minimise le coût total des colonnes sélectionnées. Les contraintes (2.9) assurent que chaque ligne soit couverte exactement par une colonne et les contraintes (2.10) indiquent que les variables x_j sont binaires.

Étant donné que nous n'avons pas trouvé dans la littérature une preuve que le SPP est NP-complet, nous présentons la démonstration dans la section suivante.

2.2.1 Complexité du SPP

Afin de transformer le SPP en problème de décision, nous introduisons une constante K et nous posons le problème (P) suivant :

Existe-t-il un vecteur binaire $x = (x_j)_{j\in J}$ tel que les contraintes suivantes :
$\sum_{j\in J} c_j x_j \leq K$ et $\sum_{j\in J} a_{ij} x_j = 1$ soient vérifiées ?

Pour démonter que (P) est un problème NP-complet, nous devons démontrer qu'il est NP-difficile et qu'il appartient à la classe NP.

Garey et Johnson (1979) ont défini le problème $[SP2]$ qu'ils ont nommé $X3C$. Ce dernier peut être formulé comme le problème de trouver un vecteur binaire $x = (x_j)_{j\in J}$ tel que $Ax = b$, où toutes les composantes de b sont égales à 1 et $A = (a_{ij})_{1\leq i\leq m, 1\leq j\leq n}$ est une matrice binaire contenant exactement trois 1 dans chaque colonne, c'est-à-dire que pour tout $j = 1, \dots n$, la propriété $\sum_{i=1}^{m} a_{ij} = 3$ est vérifiée.

Ainsi déterminer si $Ax = b$ possède une solution ou non est équivalent à déterminer si $\left\{ \sum_{j\in J} x_j \leq n, Ax = b \right\}$ possède une solution ou non.

Ce dernier problème est la version « problème de décision » du problème de partitionnement

$$\begin{cases} \min \quad z = \sum_{j\in J} x_j \\ s.c. \quad Ax = b \\ \qquad x_j \in \{0,1\} \qquad \forall j \in J \end{cases}$$

Ceci prouve que le problème $[SP2]$ peut être réduit à (P) en temps polynomial. Par conséquent, le problème (P) est NP-difficile (pour la définition de NP-difficile voir Garey et Johnson (1979)).

De plus, étant donné un vecteur binaire $x = (x_j)_{j\in J}$ quelconque, il est possible de déterminer en temps polynomial si ce dernier satisfait ou non les contraintes $\sum_{j\in J} c_j x_j \leq K$ et $\sum_{j\in J} a_{ij} x_j = 1$. Par conséquent, le problème (P) appartient à la classe NP.

Nous avons montré que :

- le problème $[SP2]$ peut être réduit au problème (P), et
- le problème (P) appartient à la classe NP.

Nous pouvons donc affirmer que le problème (P) est un problème NP-complet.

Le SPP est l'un des plus anciens et importants problèmes traités en recherche opérationnelle. Malgré la simplicité du modèle mathématique, le SPP est un problème NP-complet. Nous retrouvons dans la littérature plusieurs problèmes pratiques qui sont formulés comme des problèmes de partitionnement. Le problème le plus étudié est celui de la planification d'horaires d'équipages.

Balas et Padberg (1976) ont présenté une revue de littérature regroupant les résultats théoriques et les méthodes de résolution des problèmes de partitionnement et de couverture d'ensemble. Hoffman et Padberg (1993) ont proposé un algorithme de type branch-and-cut pour la résolution des problèmes de partitionnement provenant de l'industrie aérienne. Les plans coupants ajoutés sont les inégalités de clique, les inégalités de cycle impair et les inégalités complémentaires des inégalités de cycle impair. Chu et Beasley (1998) ont proposé un algorithme génétique pour la résolution du SPP. Thompson (2002) a proposé un algorithme basé sur la méthode du simplexe intégrale pour la résolution des problèmes d'optimisation. Elhallaoui *et al.* (2005) ont proposé une méthode de génération de colonnes, qui permet de réduire le nombre de contraintes du problème maître, en agrégeant certaines contraintes. Lewis *et al.* (2008) ont reformulé le SPP comme un problème quadratique, et ont utilisé une heuristique de recherche tabou pour le résoudre. La plupart des méthodes proposées pour la résolution des problèmes de partitionnement avec des contraintes supplémentaire sont soit de type « branch-and-price-and-cut » soit des heuristiques. Nous n'avons pas réussi à trouver des instances de SPP pur, sauf les instances utilisées par Hoffman et Padberg (1993). Lewis *et al.* (2008) ont proposé un générateur aléatoire d'instances de SPP.

2.2.2 Les inégalités valides existantes

Les plus importantes inégalités introduites pour le SPP sont les inégalités de clique et les inégalités de cycle impair. Les autres inégalités valides sont complémentaires aux deux premières. Nous retrouvons plus de détails dans Balas et Padberg (1976).

Afin d'identifier ces deux classes d'inégalités valides, un graphe simple non orienté $G = (V, E)$ est associé au SPP, où l'ensemble des sommets V représente l'ensemble des colonnes, et il existe une arête $(j, j') \in E$ si et seulement si les colonnes j et j' ont au moins une ligne en commun, c'est-à-dire qu'il existe au moins une ligne $i \in I$ telle que $a_{ij} = a_{ij'} = 1$.

Les inégalités de clique

Définition 2.2.1 *Soit $K = (S, E(S))$ un sous-graphe induit de $G = (V, E)$, où $|S| = k$. On dit que K est une clique de cardinalité k si et seulement s'il existe une arête dans $E(S)$ reliant toute paire de sommets de S.*

Théorème 2.2.2 *(Padberg (1973)) Toute solution entière du SPP vérifie l'inégalité de clique*

$$\sum_{j \in S} x_j \leq 1 \tag{2.11}$$

où $K = (S, E(S))$ est une clique.

Le théorème 2.2.2 a été démontré par Padberg (1973). L'auteur a démontré que si la clique est maximale, alors l'inégalité valide correspondante est une facette pour le SPP. Nous retrouvons dans la littérature des heuristiques pour la séparation des inégalités de clique, mais il n'existe pas encore un algorithme polynomial pour la séparation d'une clique de cardinalité donnée.

Les inégalités de cycle impair

Définition 2.2.3 *Un cycle impair de $G = (V, E)$ est un ensemble de sommets $C = (v_1, v_2, ..., v_{2k+1})$, $\forall k \geq 1$, tel que chaque sommet v_i est relié par une arête de E au sommet v_{i+1}, $\forall i = 1, 2, ..., 2k + 1$, avec $v_{2k+2} = v_1$. De plus, s'il n'existe pas d'autres arêtes reliant deux sommets de C, alors C est un cycle impair sans corde (ou trou impair).*

Théorème 2.2.4 *Toute solution entière du SPP vérifie l'inégalité de cycle impair*

$$\sum_{j \in C} x_j \leq \left\lfloor \frac{|C|}{2} \right\rfloor \tag{2.12}$$

où C est un cycle impair sans corde (ou trou imapir).

Les inégalités de cycle impair sont utilisées pour les problèmes de partitionnement et de couverture d'ensemble. Nemhauser et Sigismondi (1992) ont proposé un algorithme polynomial pour séparer ces inégalités pour le problème de stable. Hoffman et Padberg (1993) ont proposé le même algorithme pour la séparation des inégalités de cycle impair pour le SPP, mais les auteurs ont implémenté un algorithme glouton pour la séparation de ces inégalités. Nous retrouvons dans la littérature d'autres heuristiques pour la séparation des inégalités de cycle impair.

Méthode de séparation des trous impairs pour le problème de stable (Nemhauser et Sigismondi (1992))

Soit un graphe $G = (V, E)$ sans boucles ni arêtes multiples, où V est l'ensemble des sommets et E représente l'ensemble des arêtes de G. Un stable de G est un sous-ensemble de

sommets deux à deux non adjacents ; en d'autres termes chaque arête du graphe a au plus une extrémité dans le stable. Soit p le vecteur des poids associés aux sommets de G ; le poids d'un sous-ensemble de sommets est la somme des poids de tous ses sommets. Le problème du stable de poids maximum consiste à trouver un stable S tel que la somme des poids des sommets de S soit maximale.

Il est facile de formuler mathématiquement le problème. Soit la variable binaire x_v telle que

$$x_v = \begin{cases} 1 & \text{si le sommet } v \text{ appartient au stable} \\ 0 & \text{sinon.} \end{cases}$$

Le problème du stable s'écrit sous forme de programme linéaire en nombres entiers comme suit :

$$(Stable) \begin{cases} \max \quad z \quad = \quad \sum_{v \in V} p_v x_v \\ \\ \text{s.c.} \qquad x_u + x_v \leq 1 \quad \forall (u,v) \in E \\ \\ \qquad\qquad x_v \in \{0,1\} \;\; \forall v \in V. \end{cases}$$

La solution optimale fractionnaire de la relaxation linéaire du problème traité est dénotée $x = (x_i)_{i \in V}$. À chaque arête $(i,j) \in E$, nous associons le coût $c_{ij} = 1 - x_i - x_j \geq 0$. Nous construisons un graphe biparti $G' = (V, V', E')$, où G' contient les arêtes (i,j') et (j,i') si et seulement si (i,j) est une arête de G. Les coûts associés à chaque arête sont $c_{ij'}$ et $c_{ji'}$, où $c_{ij'} = c_{ji'} = c_{ij}$. Notons qu'un chemin impair dans G' de v_0 à v'_0 est de la forme $(v_0, v'_1, v_2, v'_3, ..., v_{2k}, v'_0)$, correspondant au cycle impair $(v_0, v_1, v_2, v_3, ..., v_{2k}, v_0)$ dans G. Ce dernier est un cycle comprenant $2k+1$ sommets. Les poids des chemins dans G et G' sont identiques et égaux à

$$2k + 1 - 2 \sum_{j=0}^{2k} x_{v_j}. \tag{2.13}$$

Il suit que le chemin de poids minimum de v_0 à v'_0 fournit un cycle impair C dans G qui maximise $\sum_{v \in C} x_v - \frac{|C|-1}{2}$ parmi tous les cycles impairs de G contenant v_0, et

$$\sum_{v \in C} x_v - \frac{|C|-1}{2} > 0 \tag{2.14}$$

si et seulement si le poids du chemin correspondant dans G' est inférieur à 1 (c'est-à-dire que $2k + 1 - 2\sum_{j=0}^{2k} x_{v_j} < 1$).

Supposons que nous avons trouvé un cycle impair de poids minimum C contenant v_0 et tel que $\sum_{v \in C} x_v > \frac{|C|-1}{2}$. Trois cas sont possibles :

- si $|C| = 3$, le cycle C est un triangle et nous avons obtenu une contrainte violée,
- si $|C| \geq 5$ et C est un cycle impair sans corde ; nous avons une contrainte violée aussi,
- si $|C| \geq 5$ et C est un cycle avec une ou plusieurs cordes. on remarque que chaque corde partitionne le cycle C en un cycle impair C_1 et un chemin P avec un nombre pair de sommets. Il suit que $\sum_{v \in C_1} x_v + \sum_{v \in P} x_v = \sum_{v \in C} x_v > \frac{|C|-1}{2} = \frac{|C_1|-1}{2} + \frac{|P|}{2}$.

Dans le problème du stable, nous avons les contraintes sur les arêtes $x_u + x_v \leq 1$, pour tout $(u, v) \in E$. Si nous faisons la somme des contraintes sur les arêtes pour une famille $\mathcal{F}$ d'arêtes partitionnant P, nous obtenons

$$\sum_{v \in P} x_v \leq \frac{|P|}{2} \iff \frac{|P|}{2} - \sum_{v \in P} x_v \geq 0, \tag{2.15}$$

et donc $\sum_{v \in C_1} x_v > \frac{|C_1|-1}{2} + \frac{|P|}{2} - \sum_{v \in P} x_v > \frac{|C_1|-1}{2}$.

Nous avons obtenu un cycle impair C_1 qui satisfait la condition (2.14). Nous appliquons la même procédure jusqu'à ce que nous obtenions un trou impair. Notons que l'inégalité correspondant à C est redondante si C contient une corde.

Algorithme de recherche de plus court chemin

Le meilleur algorithme pour résoudre le problème du plus court chemin est l'algorithme de Dijkstra, élaboré en 1959, qui détermine le plus court chemin d'un sommet source à tous les autres sommets d'un graphe orienté et pondéré $G = (V, A)$, dans le cas où les poids de tous les arcs du graphe sont positifs ou nuls. L'algorithme de Dijkstra gère un ensemble de sommets S, dont les longueurs finales des plus courts chemins à partir d'une source unique ont été calculées. Soit s la source et $w(u, v) \geq 0$ le poids de l'arc (u, v) ; $w : V \to \mathbb{R}$ est la fonction de pondération associée au graphe G. À chaque itération, l'algorithme 2.1 (tiré de Cormen et *al.* 2002) choisit un sommet u de V qui n'est pas dans S et dont la distance présumée à la source est minimale, ajoute u à S, ensuite relâche tous les arcs adjacents à u. Le rôle de la procédure *Relâcher*(u, v, w) (voir l'algorithme 2.2) consiste à tester s'il est possible d'améliorer le plus court chemin vers v, en passant par u. Si c'est vrai, $d(v)$, qui

représente la longueur du plus court chemin de la source à v trouvé jusqu'ici, est actualisée, ainsi que le prédécesseur de v qui deviendra u.

Algorithme 2.1 Dijkstra(G, w, s)

```
1: S := ∅
2: F := V (F est une file de priorité)
3: d(s) := 0, d(u) := +∞ pour tout u ≠ s
4: Prédécesseur(u) := NULL pour tout u ≠ s
5: tant que F ≠ ∅ faire
6:    min := Extraire_Min(F)
7:    S := S ∪ {min}
8:    pour chaque sommet v adjacent à min faire
9:       Relâcher(min, v, w)
10:   finpour
11: fintant que
```

Algorithme 2.2 *Relâcher(u,v,w)*

```
1: si d(v) > d(u) + w(u, v) alors
2:    d(v) := d(u) + w(u, v)
3:    Prédécesseur(v) := u
4: finsi
```

L'algorithme de Dijkstra (comme présenté dans Cormen et *al.* 2002) fait appel à l'opération *Insérer* (insertion des sommets de V dans la file F) une seule fois pour chaque sommet ; l'opération $Extraire\_Min$ est aussi appelée une fois par sommet. Par contre, l'opération *Relâcher(u,v,w)* est appelée une fois pour chaque arc de la liste d'adjacence de chaque sommet ; donc il y a au plus $|A|$ appels de *Relâcher(u,v,w)*.

Nous avons donc un coût total de $|V|$*Insérer* + $|V|Extraire\_Min$ + $|A|$Relâcher. La complexité de l'algorithme dépend de l'implémentation de la file de priorité.

Nous avons implémenté l'algorithme en utilisant les tas binaires, où chaque opération s'exécute en $O(\log(|V|))$. Donc, l'agorithme de Dijkstra s'exécute en un temps $O((|A| + |V|)\log(|V|))$. Notons aussi qu'il est possible d'utiliser les tas de Fibonacci qui permettent d'obtenir un temps d'exécution $O(|A|\log(|A|) + |V|)$.

2.3 Problème du transversal de circuits de cardinalité minimale (MINFVS)

Le MINFVS consiste à déterminer le plus petit sous-ensemble de sommets tel que si nous supprimons ces derniers, le graphe devient acyclique. Notons qu'il existe plusieurs versions

du MINFVS : le graphe peut être orienté ou non orienté et les sommets (arcs) peuvent être pondérés ou non.

Considérons un graphe orienté $G = (V, A)$, où V est l'ensemble des sommets et A l'ensemble des arcs. Un circuit dans G est une suite $(v_o, v_1, ..., v_{l-1})$ de sommets différents de G, (où « + » indique une addition modolu l) telle que (v_i, v_{i+1}) est un arc de G, pour tout $i = 0, 1, ..., l-1$, (en supposant $v_l = v_0$). Un circuit $C = (v_o, v_1, ..., v_{l-1})$ est sans corde s'il n'existe pas un arc (v_i, v_j) où $j \neq i+1$ modulo l. Le graphe G est dit acyclique s'il ne contient aucun circuit (ou circuit sans corde). Le sous-ensemble $U \subseteq V$ est un transversal de circuits de G si le sous-graphe induit par l'ensemble des sommets $V \setminus U$ est acyclique, c'est-à-dire l'intersection de U avec tout circuit de G est non vide. À chaque sommet $i \in V$ nous associons une variable binaire x_i qui prend la valeur 1 si et seulement si le sommet i appartient à l'ensemble U et 0 sinon. Soit $\mathcal{C}$ l'ensemble de tous les circuits sans corde du graphe G. Le MINFVS se formule comme un programme linéaire en nombres entiers.

$$(IP_{MFVS}) \begin{cases} \min & \sum_{i \in V} x_i & & (2.16) \\ \text{s.c.} & \sum_{i \in C} x_i \geq 1 & \forall C \in \mathcal{C} & (2.17) \\ & x_i \in \{0, 1\} & \forall i \in V & (2.18) \end{cases}$$

L'objectif (2.16) minimise la cardinalité de l'ensemble U. Les contraintes (2.17) assurent que chaque circuit du graphe soit couvert par au moins un sommet appartenant à U, et les contraintes (2.18) indiquent que les variables sont binaires.

Le MINFVS est l'un des plus anciens problème de l'optimisation combinatoire. Karp (1972) a montré que le MINFVS est un problème NP-complet. Festa et al. (1999) présentent une revue de littérature sur les différentes versions du MINFVS. Grötschel *et al.* (1985) et Funke et Reinelt (1996) ont étudié une des variantes du MINFVS en utilisant des méthodes polyédrales. Even *et al.* (1998) ont proposé des algorithmes pour la détermination de solutions approchées de plusieurs variantes de MINFVS. Dehne *et al.* (2007) et Chen *et al.* (2008) ont conçu des algorithmes pour construire un MINFVS avec au plus k sommets ou montrer qu'il n'existe pas dans des graphes orientés et non orientés. Orenstein *et al.* (1995) ont étudié le MINFVS pour des problèmes de conception et de tests des circuits éléctroniques. Les auteurs ont utilisé des techniques de réduction et de partitionnement de graphes. Levy et Low (1988) et Lin et Jou (2000) ont introduit plusieurs techniques permettant de réduire la taille du graphe orienté. Lin et Jou (2000) ont proposé un algorithme exact basé sur une méthode de séparation et évaluation progressive pour la résolution des instances du MINFVS.

2.3.1 Les inégalités de Chvátal-Gomory

Parmi les inégalités valides pour tout problème en nombres entiers, nous retrouvons les inégalités de Chvátal-Gomory. Considérons l'ensemble des contraintes $S = \{x \in \mathbb{Z}^n \mid Ax \geq b\}$ avec $A \in \mathbb{Z}^{m \times n}$ et $b \in \mathbb{Z}^m$. Pour tout $u \in \mathbb{R}^m$ tel que $u \geq 0$, l'inégalité

$$\lceil uA \rceil x \geq \lceil ub \rceil \tag{2.19}$$

est valide pour l'ensemble des contraintes S, et est nommée inégalité de Chvátal-Gomory.

Gomory (1963) a proposé une méthode de séparation pour déterminer un plan coupant séparant un point extrême du polyèdre de la relaxation de S qui n'est pas un point entier. Caprara et Fischetti (1996) ont étudié une classe des inégalités de Chvátal-Gomory nommée $\{0, \frac{1}{2}\}$-Chvátal-Gomory, obtenue lorsque $u \in \{0, \frac{1}{2}\}^m$. Les auteurs ont montré que le problème de séparation de cette classe est NP-complet dans le cas général. Les auteurs ont proposé aussi un algorithme polynomial pour la séparation des inégalités de Chvátal-Gomory obtenue lorsque $u = \frac{k-1}{k}$, $\forall k \in \mathbb{N}^*$, pour le problème de voyageur de commerce. Eisenbrand (1999) a démontré que le problème de séparation des inégalités de Chvátal-Gomory générales est NP-difficile. Nous retrouvons dans la littérature des méthodes heuristiques pour la séparation des inégalités de Chvátal-Gomory.

Nous n'avons présenté dans ce chapitre que les inégalités valides que nous utilisons dans les chapitres suivants. Nous retrouvons dans Bonami (2003) une revue de littérature sur les plus importantes inégalités valides pour les programmes en nombres entiers et mixtes généraux.

2.4 Méthode d'élimination de variables

Afin d'accélérer la résolution de la relaxation linéaire à chaque noeud de l'arbre de branchement, il est possible d'éliminer (ou fixer) des variables sous certaines conditions. La technique est basée sur les valeurs des coûts réduits des variables. Dans cette section, nous présentons deux méthodes d'élimination de variables : la première est directe et la deuxième est bidirectionnelle.

Bianco *et al.* (1994) ont utilisé la méthode directe afin de réduire le nombre de variables dans la formulation de partitionnement d'ensemble pour le MDVSP. Hadjar *et al.* (2006) ont utilisé cette méthode à chaque noeud de l'arbre de branchement, où la relaxation linéaire du problème est résolue par la génération de colonnes. Cette stratégie a permis aux auteurs de supprimer, en moyenne, 90% des arcs du graphe associé au problème auxiliaire de la génération de colonnes.

Lorsque la génération de colonnes est utilisée pour résoudre un problème, la stratégie

d'élimination de variables n'est pas appliquée directement au problème maître, mais au problème auxiliaire associé. Par conséquent, pour fixer une variable, il suffit de supprimer l'arc correspondant dans le réseau sous-jacent lors de la résolution du problème auxiliaire, mais l'inconvénient est que les valeurs des variables duales ne sont pas obtenues directement.

Nous retrouvons dans la littérature deux méthodes basées sur la décomposition de Dantzig-Wolfe : la première est proposée par Poggi de Aragão *et al.* (2003), qui ont suggéré d'ajouter une contrainte suplémentaire au problème maître, reliant les variables de ce dernier aux variables du problème auxiliaire. Ainsi les coûts réduits peuvent être récupérés directement de la solution du problème maître. La deuxième méthode consiste à appliquer le principe de Walker (1969), qui indique que si le problème auxiliaire peut être formulé comme un programme linéaire, alors la solution optimale du programme linéaire global peut être obtenue à l'aide de la solution optimale duale du problème maître et la solution optimale duale du problème auxiliaire.

Une récente étude d'Irnich *et al.* (2010) montre qu'il est possible de calculer la meilleure valeur possible du coût réduit en utilisant une recherche bidirectionnelle. En effet, étant donné que les coûts réduits dépendent de la solution duale, il peut exister plusieurs coûts réduits pour la même variable à cause de l'existance de plusieurs solutions duales. En utilisant un modèle issu de la décomposition de Dantzig-Wolfe et dans le contexte de génération de colonnes, les auteurs ont montré qu'il est possible d'éliminer un grand nombre d'arcs, si le coût réduit d'un arc (i, j) du réseau est fonction des valeurs du plus court chemin de la source à i, du plus court chemin de j au puits et du plus court chemin de la source au puits. Irnich *et al.* (2010) ont appliqué cette technique au VRPTW, où le problème auxiliaire de la méthode de génération de colonnes est un problème de plus court chemin avec contraintes de ressources. Les auteurs ont montré que la recherche bidirectionnelle fournit de meilleurs résultats (en termes de nombre d'arcs éliminés et de temps de résolution) que la recherche monodirectionnelle ; dans le pire cas, les résultats sont identiques.

2.4.1 Méthode directe

Considérons le programme linéaire en nombres entiers suivant, où toutes les données sont entières.

$$PLE \begin{cases} \min \quad z \;=\; cx \\ \quad \text{s.c.} \qquad Ax = b \\ \qquad\qquad\quad x \in \mathbb{Z}^n, \;\; x \geq 0 \end{cases}$$

Dénotons par PL la relaxation linéaire de PLE.

Proposition 2.4.1 *Soit $\bar{x}$ une solution réalisable de PLE, x' une solution réalisable de PL et y' une solution réalisable du dual de PL. La variable x_i^0 prend la valeur 0 dans toute solution réalisable x^0 de PLE telle que $cx^0 < c\bar{x}$, si le coût réduit de la variable x_i^0 par rapport à y' est supérieur ou égal à $c\bar{x} - y'b$.*

La proposition 2.4.1 a été prouvée par Hadjar *et al.* (2006) et a été énoncée sous une forme plus restreinte par Nemhauser et Wosley (1988 p.389).

2.4.2 Méthode bidirectionnelle

Irnich *et al.* (2010) ont reformulé la proposition 2.4.1 en utilisant la formulation d'un programme linéaire en nombres entiers dont les contraintes se séparent en deux groupes comme suit :

$$PLE \begin{cases} \min \quad z = cx & (2.20) \\ \text{s.c.} \quad \text{Ax} = b & (2.21) \\ \qquad Dx = d & (2.22) \\ \qquad x \in \mathbb{Z}^n, \;\; x \geq 0 & (2.23) \end{cases}$$

Soit (π, ρ) une solution réalisable duale de PL (la relaxation linéaire de PLE), $r(\pi, \rho)^T = c^T - \pi A - \rho D$ le vecteur des coûts réduits des variables, et UB une borne supérieure sur la valeur optimale z^* de PLE.

Selon la proposition 2.4.1, si $r_{ij}(\pi, \rho) > UB - \pi b - \rho d$, alors on a $x_{ij} = 0$ dans toute solution optimale de PLE. Ceci peut s'écrire : $lb_{ij}(\pi, \rho) = r_{ij}(\pi, \rho) + \pi b + \rho d > UB$, où lb_{ij} est une borne inférieure pour la valeur de toute solution réalisable de PLE avec $x_{ij} \geq 1$.

Irnich *et al.* (2010) ont étudié le cas où le problème auxiliaire de la génération de colonnes est un problème de plus court chemin (PPCC), étant donné que ce dernier peut se formuler comme un programme linéaire tant que le réseau associé au problème ne contient pas de cycles de longueur négative. Pour le MDVSP, Ribeiro et Soumis (1994) ont donné la formulation du PPCC, où la matrice D est égale à la matrice d'incidence I^N du réseau $N = (V, \mathcal{A}, c_{ij} - (\pi A)_{ij})$ et $d = e_s - e_t$, où e_s et e_t sont les vecteurs unitaires correspondant aux noeuds source et puits, respectivement.

La conséquence de l'application de la génération de colonnes est que pour tout $\pi \geq 0$, le coût réduit de chaque variable x_{ij} peut se calculer comme suit :

$$r_{ij}(\pi, -\ell) = c_{ij} - (\pi A)_{ij} + \ell_i - \ell_j,$$

où $(\rho_i^*)_{i \in V} = (-\ell_i)_{i \in V}$ est la distance du plus court chemin de la source s à tout noeud $i \in V$.

Irnich *et al* (2010) ont montré que, dans un réseau acyclique,

$$\bar{r}_{ij}(\pi) = r_{ij}(\pi, \rho^*) = c_{ij} - (\pi A)_{ij} + \ell_i^{\rightarrow} + \ell_j^{\leftarrow} - \ell_t^{\rightarrow} \tag{2.24}$$

et

$$\bar{lb}_{ij}(\pi) = \bar{r}_{ij}(\pi) + \pi b + \ell_t^{\rightarrow}, \tag{2.25}$$

où :

- $\bar{r}_{ij}(\pi)$ est la plus grande valeur du coût réduit de la variable x_{ij} par rapport à la solution duale π,
- $\bar{lb}_{ij}(\pi) = \max_\rho lb_{ij}(\pi, \rho)$, pour $\pi > 0$, est la plus grande borne inférieure,
- $\ell_t^{\rightarrow} = \rho^* d$ est la longueur du plus court chemin de la source s au puits t,
- $\ell_i^{\rightarrow}$ est la longueur du plus court chemin de la source s à i
- $\ell_j^{\leftarrow}$ est la longueur du plus court chemin du puits t à j en inversant le sens des arcs.

Le résultat trouvé peut s'appliquer même si la génération de colonnes n'est pas utilisée, c'est-à-dire qu'au lieu d'utiliser un seul ρ. En effet, nous calculons un plus court chemin avant et un autre arrière pour obtenir la borne inférieure $lb_{ij}(\pi)$ la plus élevée pour chaque arc, permettant ainsi d'éliminer le plus grand nombre d'arcs possible.

CHAPITRE 3

INÉGALITÉS VALIDES ET ALGORITHMES DE SÉPARATION POUR LE MDVSP

3.1 Introduction

Hadjar *et al.* (2006) ont introduit une nouvelle classe d'inégalités valides propres au MDVSP. Les auteurs ont utilisé la notion de conflit entre les arcs du multigraphe associé au MDVSP, afin de définir la strucure de sous-multigraphe épineux. Un tel sous-multigraphe permet d'écrire les inégalités valides du MDVSP. Les auteurs ont utilisé une procédure d'énumération pour la séparation des cycles impairs et comme cette méthode consomme beaucoup de temps, les auteurs ont limité la taille des cycles à 7. Ils ont résolu le MDVSP à l'aide d'un algorithme combinant la méthode directe d'élimination de variables, la méthode de séparation et évaluation progressive, la méthode de génération de plans coupants et la méthode de génération de colonnes.

Dans le présent chapitre, nous proposons de résoudre le MDVSP en utilisant une méthode combinant l'élimination de variables en deux phases, la méthode de séparation et évaluation progressive et la méthode de génération de plans coupants. Nous proposons de générer des plans coupants en séparant des structures qui ne sont pas forcément des cycles impairs.

Dans la section 3.2, nous présentons une généralisation de la notion de sous-multigraphe épineux définie par Hadjar *et al.* (2006). Dans la section 3.3, nous montrons qu'il existe une relation, sous certaines conditions, entre le problème du stable et le MDVSP en utilisant le graphe de conflit du MDVSP. Nous adaptons l'algorithme de Nemhauser et Sigismondi (1992) pour déterminer une classe d'inégalités valides que nous nommons inégalités de cycle impair conflictuel et nous décrivons une méthode de séparation. Dans la section 3.4, nous présentons un algorithme glouton pour séparer les cycles de longueur 3, qui sont aussi des cliques de cardinalité 3. Dans la section 3.5, nous définissons une correspondance entre le MDVSP et le problème de couplage en utilisant une matrice des sous-chemins. Nous utilisons les inégalités d'Edmonds (1965) afin d'identifier une autre classe d'inégalités valides que nous nommons des inégalités de coupe impaire et nous décrivons une méthode de séparation. Dans la section 3.7, nous décrivons un algorithme de type « branch-and-cut » pour la résolution du MDVSP et les stratégies testées. Dans la section 3.8, nous présentons les résultats des tests en utilisant des instances aléatoires et nous terminons par une conclusion dans la section 3.9.

3.2 Les inégalités valides pour le MDVSP

Il existe une forte ressemblance entre le problème de couplage et le MDVSP. Un couplage est un ensemble d'arêtes deux à deux non adjacentes. Par conséquent, deux arêtes adjacentes sont toujours en conflit, mais ceci n'est pas toujours vrai pour le MDVSP. Par contre, il est possible de trouver une structure conflictuelle, dans laquelle deux sous-chemins adjacents sont en conflit. Edmonds (1965) a montré que l'enveloppe convexe du problème de couplage est définie par certains types d'inégalités linéaires (« Blossom inequalities »). Padberg et Rao (1982) ont montré qu'il existe une inégalité d'Edmonds (ou « blossom inequality ») violée si et seulement si le graphe associé à la solution fractionnaire contient une coupe impaire de capacité strictement inférieure à 1. Notons aussi qu'un couplage dans un graphe donné correspond à un stable dans son graphe adjoint puisqu'un sommet dans ce dernier représente une arête dans le graphe initial. Par conséquent, il existe aussi une relation entre le MDVSP et le problème de stable. Nemhauser et Sigismondi (1992) ont proposé une méthode de séparation des inégalités de trou impair pour le problème de stable. Dans cette section, nous décrirons une relation entre le MDVSP et les problèmes de couplage et de stable et sous quelles conditions elle est valide.

Avant de présenter la généralisation de la notion de sous-multigraphe épineux et les détails des méthodes de séparation, nous introduisons la définition 3.2.1. Cette dernière nous permet d'alléger les définitions subséquentes.

Définition 3.2.1 *Soit $G^* = (S, F)$ un sous-multigraphe orienté, où S ne contient aucun dépôt,*

- *Un noeud $i \in S$ est dit noeud source si $\delta_F^-(i) = \emptyset$.*
- *Un noeud $i \in S$ est dit noeud puits si $\delta_F^+(i) = \emptyset$.*
- *Un noeud $i \in S$ est dit noeud ordinaire si $\delta_F^-(i) \neq \emptyset$, $\delta_F^+(i) \neq \emptyset$ et $Col(\delta_F^-(i)) \bigcap Col(\delta_F^+(i)) = \emptyset$.*
- *Un noeud $i \in S$ est dit noeud de conflit s'il est une source, un puits ou un noeud ordinaire.*
- *Un noeud $i \in S$ est dit noeud intermédiaire si $\delta_F^-(i) \neq \emptyset$, $\delta_F^+(i) \neq \emptyset$ et $Col(\delta_F^-(i)) \bigcap Col(\delta_F^+(i)) \neq \emptyset$.*

Considérons la solution fractionnaire illustrée par le multigraphe $G = (V, A)$ associé au MDVPS (voir la figure 3.1). Soit $H = (V', A')$ le sous-multigraphe partiel (voir la figure 3.2 a)) obtenu en supprimant les sommets dépôts $D_k \in V$, $\forall k \in K$, et leurs arcs adjacents dans le multigraphe G.

Nous construisons le graphe de conflit $L(H) = (A', E)$ (voir la figure 3.2 b)) du sous-multigraphe partiel $H = (V', A')$, où le sommet $T_{ij}^k \in A'$ représente l'arc $(i, j, k) \in A'$ et il

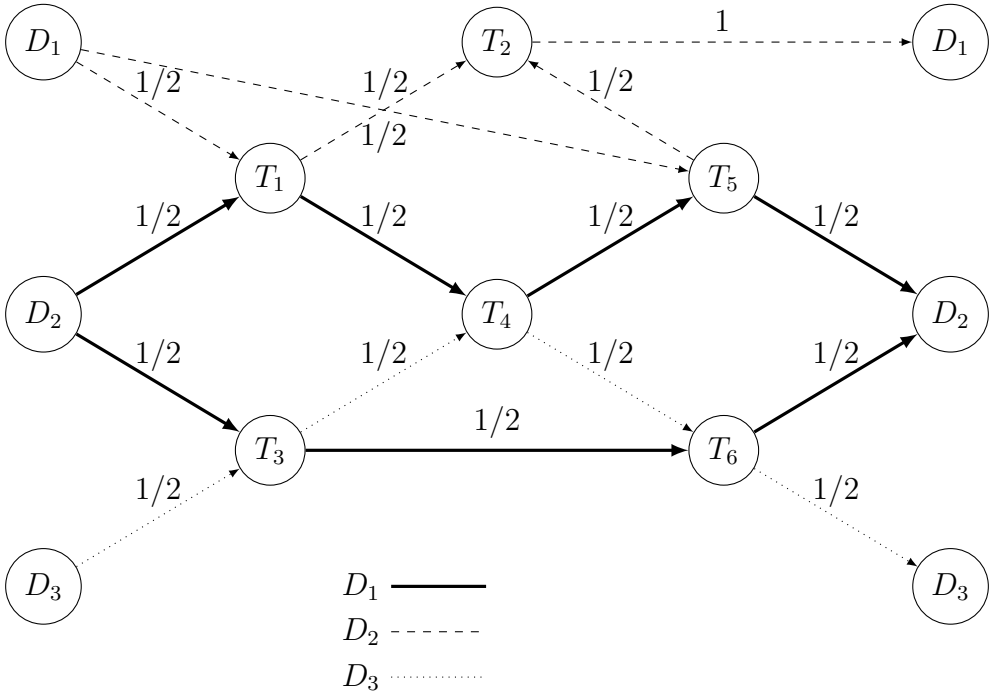

Figure 3.1 Exemple de solution fractionnaire de la relaxation linéaire du $MDVSP$

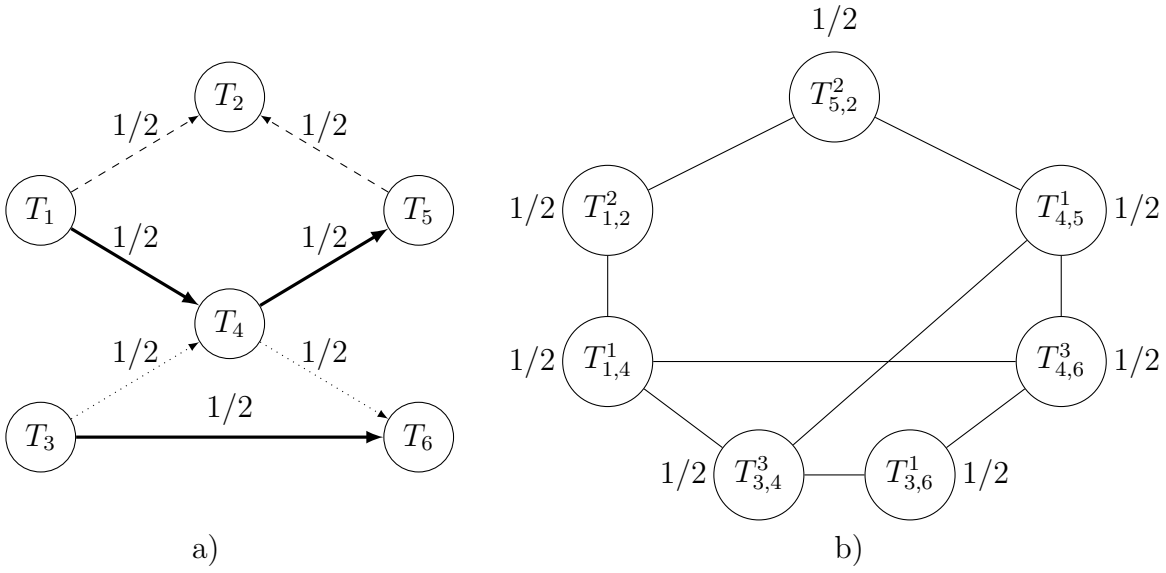

Figure 3.2 Un sous-multigraphe partiel et le graphe de conflit correspondant

existe une arête reliant les sommets T^k_{ij} et $T^{k'}_{i'j'}$ si et seulement si les arcs (i, j, k) et (i', j', k') sont en conflit. Dans le graphe de conflit $L(H) = (A', E)$, deux sommets adjacents sont

toujours en conflit. Nous constatons que chaque trou impair (cycle impair sans corde) dans $L(H) = (A', E)$ correspond à un sous-multigraphe conflictuel épineux (voir la définition 2.1.4). Comme un couplage dans un graphe correspond à un stable dans le graphe de conflit, nous utilisons l'algorithme de Nemhauser et Sigismondi (1992) pour déterminer les trous impairs dans le graphe de conflit du multigraphe partiel $H = (V', A')$ du MDVSP. Les détails de la méthode de séparation sont présentés dans la section 3.3.

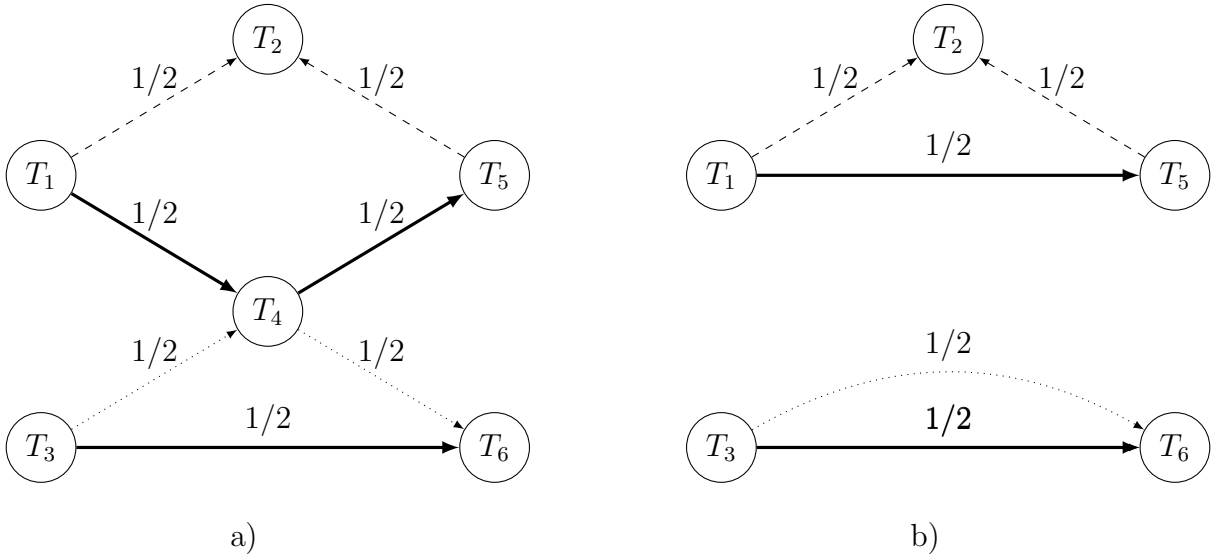

Figure 3.3 Un sous-multigraphe partiel et le sous-multigraphe des chemins correspondants

En utilisant à nouveau l'exemple de la figure 3.1, et en considérant le multigraphe partiel $H = (V', A')$, nous remarquons que, pour chaque arc entrant au sommet T_4, il existe un arc sortant de même couleur. En d'autres termes, il existe deux chemins de couleurs différentes passant par ce sommet. Nous construisons un sous-multigraphe des chemins $G'' = (V'', \mathcal{P})$ (voir la figure 3.3) du sous-multigraphe partiel $H = (V', A')$, où l'ensemble des sommets V'' représente l'ensemble des noeuds de conflit (voir la définition 3.2.1), et $\mathcal{P}$ représente l'ensemble des arcs, tel que $(i, j, k) \in \mathcal{P}$ si et seulement s'il existe un chemin de couleur k reliant les noeuds de conflit $T_i \in V''$ et $T_j \in V''$. Dans le sous-multigraphe des chemins $G'' = (V'', \mathcal{P})$, deux chemins adjacents sont toujours en conflit. Nous remarquons que chaque coupe impaire dans le sous-multigraphe $G'' = (V'', \mathcal{P})$ correspond à un sous-multigraphe conflictuel sans épine dans le sous-multigraphe partiel $H = (V', A')$. Pour la séparation de cette classe de sous-multigraphes conflictuels sans épine, nous résolvons un problème auxiliaire construit à partir de la solution fractionnaire du MDVSP. Les détails sont présentés dans la section 3.5.

Dans la définition de sous-multigraphe conflictuel épineux donné par Hadjar *et al.* (2006), pour chaque sommet intermédiaire i, l'épine est un arc appartenant à $\delta^-(i)$, mais ceci n'est

pas nécessaire pour obtenir une structure de sous-multigraphe conflictuel épineux. De plus, les auteurs imposent que l'arc représentant l'épine ne soit pas adjacent à un sommet de conflit. Cette condition aussi n'est pas obligatoire pour l'obtention d'une structure de sous-multigraphe conflictuel épineux (voir l'exemple 3.2.3). La définition 3.2.2 permet de généraliser la notion de sous-multigraphe conflictuel épineux introduite par Hadjar *et al.* (2006).

Définition 3.2.2 *Soit $G' = (S, F)$ un sous-multigraphe partiel de G. On dit que G' est un sous-multigraphe conflictuel épineux, si S peut être partitionné en S_1, S_2 et S_3 et F peut être partitionné en F_1 et F_2 vérifiant les propriétés énumérées ci-dessous. L'ensemble des arcs F_2 est l'union disjointe de la famille $\{\tau(i)\}_{i \in S_2}$, où $\tau(i)$ (l'ensemble des épines associées au sommet i) est un sous-ensemble d'arcs incident au sommet i.*

- *$S_1 \cup S_2$ ne contient aucun dépôt,*
- *$|S_1|$ est impair et au moins égal à 3,*
- *$Col(\delta^-_{F_1}(i)) \bigcap Col(\delta^+_{F_1}(i)) = \emptyset$ pour tout $i \in S_1$,*
- *$\tau(i) \subseteq \delta^+(i)$ ou $\tau(i) \subseteq \delta^-(i)$ pour tout sommet i de S_2,*
- *$Col(\tau(i)) \bigcap Col(\delta^-_{F_1}(i)) = \emptyset$ et $Col(\tau(i)) \bigcap Col(\delta^+_{F_1}(i)) = \emptyset$ pour tout $i \in S_2$, et*
- *tout arc incident à un sommet de S_3 appartient à $\tau(i)$ pour un sommet $i \in S_2$.*

En d'autres termes, F_2 est l'ensemble des épines, $F_1 = F \backslash \{F_2\}$, S_1 est l'ensemble des noeuds de conflit, S_2 est l'ensemble des noeuds intermédiaires et $S_3 = S \backslash \{S_1 \cup S_2\}$.

Exemple 3.2.3 *L'exemple de la figure 3.4 présente un sous-multigraphe conflictuel épineux $G = (S, F)$, où $S_1 = \{T_1, T_4, T_5\}$, $S_2 = \{T_2, T_3\}$ et $S_3 = \{T_6\}$,*
$F_1 = \{(T_1, T_2, 1), (T_2, T_4, 1), (T_4, T_5, 2), (T_3, T_5, 1)(T_1, T_3, 1)\}$ et $F_2 = \{(T_6, T_3, 3), (T_2, T_4, 2)\}$.

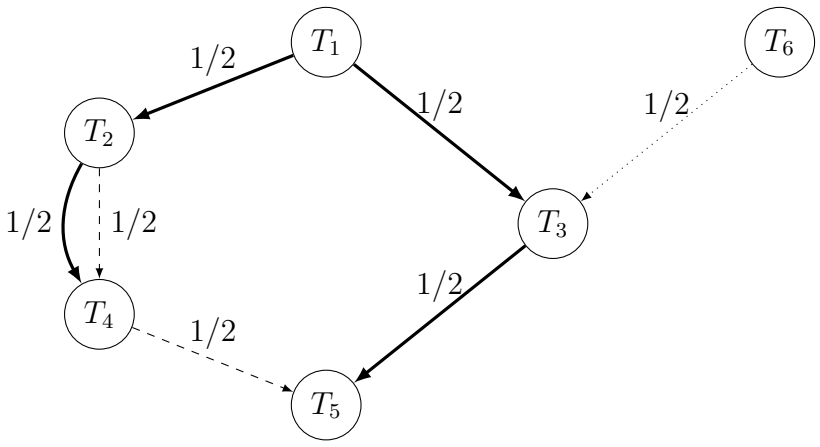

Figure 3.4 Exemple de sous-multigraphe conflictuel épineux

Exemple 3.2.4 *L'exemple de la figure 3.5 présente aussi un sous-multigraphe conflictuel épineux $G = (S, F)$, mais avec un noeud d'articulation, où $S_2 = \{T266, T294\}$, $S_3 = \{T139\}$ et $S_1 = S \backslash (S_2 \cup S_3)$ $F_2 = \{(T139, T266, 2)\}$ et $F_1 = F \backslash F_2$.*

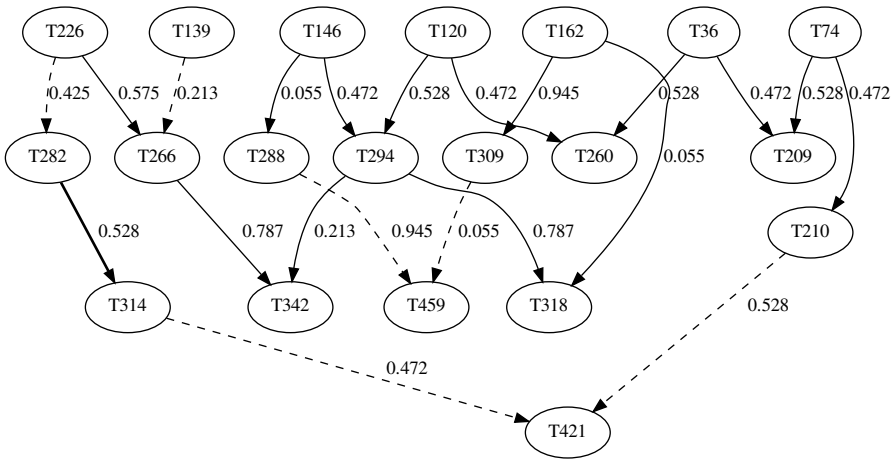

Figure 3.5 Exemple de sous-multigraphe conflictuel épineux avec un noeud d'articulation (le noeud T294)

Proposition 3.2.5 *Soit $G' = (S, F)$ un sous-multigraphe conflictuel épineux de G, où $S = S_1 \cup S_2 \cup S_3$ et $F = F_1 \cup F_2$. L'inégalité*

$$\sum_{(i,j,k) \in F} X_{i,j}^k \leq \left\lfloor \frac{|S_1|}{2} \right\rfloor + |S_2| \tag{3.1}$$

est valide pour $(MDSVP)$.

Démonstration.

Étant donné que $G' = (S, F)$ est un sous-multigraphe conflictuel épineux, alors les conditions de la définition 3.2.2 sont vérifiées. Ces dernières impliquent que :

- $\forall s \in S_1$, tous les arcs appartenant à $\delta_{F_1}(s)$ sont en conflit,
- $\forall s \in S_2$, tous les arcs appartenant à $\delta^+_{F_1}(s) \cup \tau(s)$ sont en conflit,
- $\forall s \in S_2$, tous les arcs appartenant à $\delta^-_{F_1}(s) \cup \tau(s)$ sont en conflit.

Par conséquent, les inégalités suivantes sont valides pour $(MDVSP)$:

$$\sum_{(i,j,k)\in\delta_{F_1}(s)} X_{ij}^k \leq 1 \qquad \forall s \in S_1 \tag{3.2}$$

$$\sum_{(i,j,k)\in\delta_{F_1}^+(s)} X_{ij}^k + \sum_{(i,j,k)\in\tau(s)} X_{ij}^k \leq 1 \qquad \forall s \in S_2 \tag{3.3}$$

$$\sum_{(i,j,k)\in\delta_{F_1}^-(s)} X_{ij}^k + \sum_{(i,j,k)\in\tau(s)} X_{ij}^k \leq 1 \qquad \forall s \in S_2 \tag{3.4}$$

En additionnant les inégalités (3.3) et (3.4), nous obtenons l'inégalité suivante :

$$\sum_{(i,j,k)\in\delta_{F_1}(s)} X_{ij}^k + \sum_{(i,j,k)\in\tau(s)} 2X_{ij}^k \leq 2 \qquad \forall s \in S_2 \tag{3.5}$$

Des inégalités (3.2) et (3.5), nous obtenons :

$$\sum_{s\in S_1}\sum_{(i,j,k)\in\delta_{F_1}(s)} X_{ij}^k + \sum_{s\in S_2}\Big(\sum_{(i,j,k)\in\delta_{F_1}(s)} X_{ij}^k + \sum_{(i,j,k)\in\tau(s)} 2X_{ij}^k\Big) \leq |S_1| + 2|S_2| \tag{3.6}$$

$$\Rightarrow \sum_{s\in S_1\cup S_2}\sum_{(i,j,k)\in\delta_{F_1}(s)} X_{ij}^k + \sum_{s\in S_2}\sum_{(i,j,k)\in\tau(s)} 2X_{ij}^k \leq |S_1| + 2|S_2| \tag{3.7}$$

$$\Rightarrow \sum_{(i,j,k)\in F_1} 2X_{ij}^k + \sum_{(i,j,k)\in F_2} 2X_{ij}^k \leq |S_1| + 2|S_2| \tag{3.8}$$

$$\Rightarrow \sum_{(i,j,k)\in F_1\cup F_2} X_{ij}^k \leq \frac{|S_1|}{2} + |S_2| \tag{3.9}$$

Par conséquent, l'inégalité $\sum_{(i,j,k)\in F} X_{i,j}^k \leq \left\lfloor \frac{|S_1|}{2} \right\rfloor + |S_2|$ est valide pour $(MDSVP)$. □

3.3 Séparation des inégalités de cycle impair conflictuel

La méthode de détermination des inégalités valides associées aux cycles impairs conflictuels pour le MDVSP que nous proposons consiste à utiliser la solution fractionnaire pour construire un sous-multigraphe partiel orienté, $H = (V', A')$, où l'ensemble des sommets V' représente l'ensemble des trajets $\{T_1, T_2, ..., T_n\}$, et A' l'ensemble des arcs. Il existe un arc de couleur k reliant les noeuds T_i et T_j si et seulement si le flux de l'arc (i, j, k), X_{ij}^k, est strictement positif ; ensuite, nous associons à H un graphe de conflit non orienté $L(H)$ (sous certaines conditions). À ce dernier nous appliquons la méthode de Nemhauser et Sigismondi

(1992) avec quelques modifications, afin de détecter les trous impairs qui correspondent à des sous-multigraphes épineux dans H. Nous prouvons que les sous-multigraphes épineux nous fournissent des inégalités valides, et afin d'enrichir nos inégalités valides, nous décrivons une procédure de liftage. Dans cette section nous présentons les détails de chaque étape.

3.3.1 Création des graphes

Considérons le sous-multigraphe partiel $H = (V', A')$ obtenu en supprimant les noeuds dépôts du multigraphe orienté $G = (V, A)$, c'est-à-dire que $V' = V \backslash \{D_1, D_2, \cdots, D_{|K|}\}$ et $A' = A \backslash \{(i, j, k) \mid i = D_k \text{ ou } j = D_k, \text{pour un } k \in K\}$. Le poids de chaque arc $(i, j, k) \in A'$ est égal à la valeur de la variable X_{ij}^k dans la solution fractionnaire courante.

Construisons le graphe de conflit non orienté $L(H) = (A', E)$ associé au sous-multigraphe partiel orienté $H = (V', A')$. Le graphe $L(H)$ est défini comme suit :

- les sommets de $L(H)$ sont les arcs de H,
- deux sommets de $L(H)$ sont reliés par une arête si et seulement si les deux arcs sont en conflit dans H,
- x_i est le poids du sommet $i \in A'$, qui est égal au flux de l'arc représenté par le sommet i,
- le poids de chaque arête $(i, j) \in E$ est égal à $1 - x_i - x_j$.

Remarque 3.3.1

Le poids de chaque arête est positif ou nul, puisque quels que soient les deux arcs $e_1 = (i, j, k)$ *et* $e_2 = (i', j', k')$ *en conflit, l'inégalité* $X_{ij}^k + X_{i'j'}^{k'} \leq 1$ *est valide, d'où suit la relation* $1 - X_{ij}^k - X_{i'j'}^{k'} \geq 0$. *Par conséquent,* $1 - x_{e_1} - x_{e_2} \geq 0$.

$B(H) = (A', A'', E')$ est un graphe biparti non orienté, construit à partir de $L(H) = (A', E)$, tel que :

- A' contient les sommets de $L(H)$,
- A'' contient une copie des sommets de A',
- $E' = \{(i, \mathit{copie}(j)) |\ (i, j) \in E\} \cup \{(j, \mathit{copie}(i)) |\ (i, j) \in E\}$,
- x_i (resp. $x_{\mathit{copie}(i)}$) est le poids du sommet $i \in A'$ (resp. $\mathit{copie}(i) \in A''$), avec $x_i = x_{\mathit{copie}(i)} = x_\ell$ où ℓ est le sommet correspondant dans le graphe $L(H)$,
- le poids de chaque arête $(i, \mathit{copie}(j)) \in E'$ est égal à $1 - x_i - x_j$.

À l'aide de l'algorithme de Dijkstra, pour tout sommet $i \in A'$, nous déterminons le plus court chemin de i au sommet $\mathit{copie}(i) \in A''$ dans le graphe biparti $B(H) = (A', A'', E')$. Ce plus court chemin correspond à un cycle impair C dans le graphe $L(H)$. Si C ne contient pas de corde, alors il correspond à un sous-multigraphe conflictuel épineux dans le sous-multigraphe partiel $H = (V', A')$. Sinon, nous procédons au traitement de cordes, afin de

séparer un trou impair. Si de plus le poids total des arêtes de C est strictement inférieur à 1, alors nous avons une inégalité de cycle impair conflictuel violée.

Exemple 3.3.2 *L'exemple présenté dans la figure 3.1 permet de construire les graphes H, $L(H)$ et $B(H)$ qui sont présentés dans la figure 3.6.*

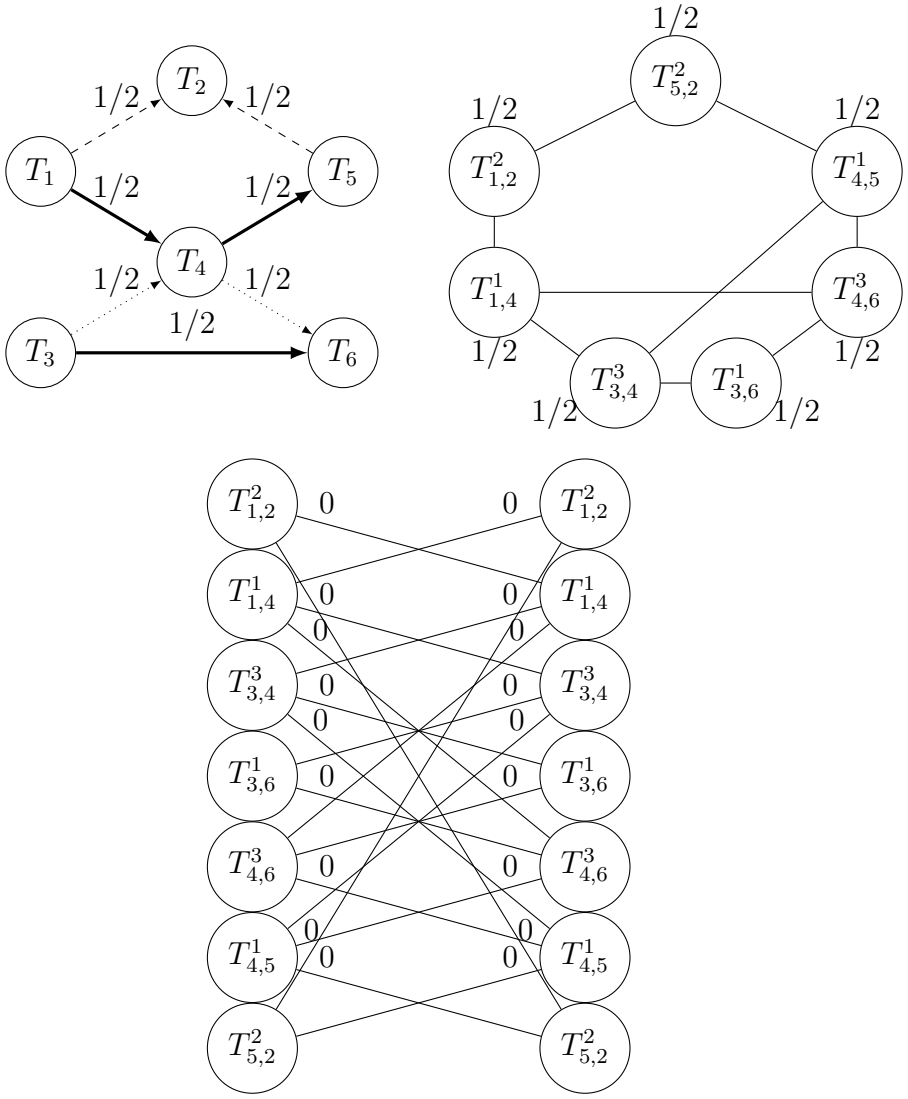

Figure 3.6 Un sous-multigraphe partiel, les graphes de conflit et biparti correspondants

3.3.2 Traitement des cordes

Soit C un cycle impair de $L(H)$, constitué des sommets $\{v_0, v_1, ..., v_{2k}, v_{2k+1} = v_0\}$; les indices des sommets de C sont ordonnés dans le sens des aiguilles d'une montre à partir du sommet v_0 (v_0 est le sommet source utilisé pour trouver le plus court chemin).

S'il existe une ou plusieurs cordes dans le cycle impair C, alors chaque corde $\{v_d, v_f\}$ partitionne C en un cycle impair C_{df} et un chemin P avec un nombre pair de sommets et un nombre impair d'arêtes égal au nombre de sommets moins 1 (voir la figure 3.7). Le cycle impair C_{df} peut contenir une ou plusieurs cordes. Pour traiter ce cas, nous introduisons la définition 3.3.3 pour définir la relation de dominance entre les cordes du même cycle impair.

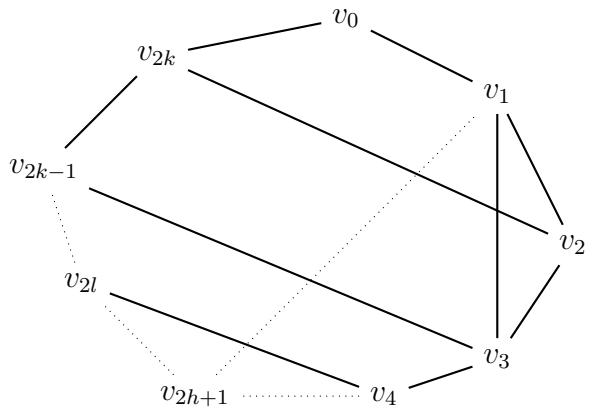

Figure 3.7 Un cycle impair contenant des cordes

Définition 3.3.3 *Soit* $\{v_{d_1}, v_{f_1}\}$ *et* $\{v_{d_2}, v_{f_2}\}$ *deux cordes dans le même cycle impair* $\{v_0, v_1, ..., v_{2k}\}$. *On dit que* $\{v_{d_2}, v_{f_2}\}$ *domine* $\{v_{d_1}, v_{f_1}\}$ *si et seulement si :*

- $d_2 \geq d_1$ *et*
- $f_2 \leq f_1$.

La figure 3.8 est une illustration de cette définition.

Proposition 3.3.4 *Supposons que le chemin* P *est composé des sommets* $\{v_{f+1}, v_{f+2}, ..., v_{2k}, v_0, v_1, ..., v_{d-1}\}$ *et qu'il existe une corde reliant les sommets* v_d *et* v_f. *Nous rappelons que le poids de chaque arête* (v_i, v_j) *de* $L(H)$ *est* $p(v_i v_j) = 1 - x_{v_i} - x_{v_j}$ *où* x_{v_i} *(resp.* x_{v_j}*) est le poids du sommet* v_i *(resp.* v_j*). Afin de faciliter l'identification des trous impairs, nous ajoutons* ϵ *(une très petite valeur positive) au poids de chaque arête du graphe* $L(H)$. *Alors*

1. *la somme des poids des arêtes formant le cycle impair* C_{df} *est strictement inférieure à la somme des poids des arêtes formant le cycle impair* C *;*
2. *le cycle* C_{df} *ne peut pas contenir le sommet source* v_0 *;*
3. *le cycle impair* C_{df} *engendré par la corde* $\{v_d, v_f\}$ *est un trou impair si et seulement s'il n'existe aucune corde qui domine* $\{v_d, v_f\}$ *;*

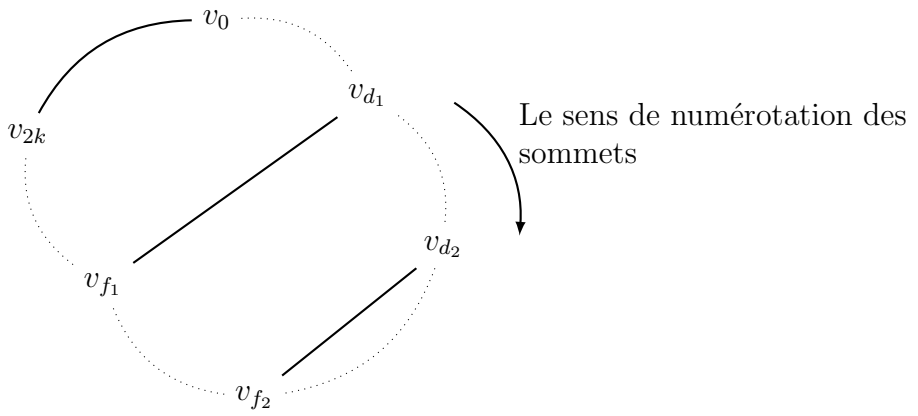

Figure 3.8 Exemple où $\{v_{d_2}, v_{f_2}\}$ domine $\{v_{d_1}, v_{f_1}\}$

4. *dans un cycle impair C, il peut exister plusieurs trous impairs, qui fournissent des inégalités valides incomparables.*

Démonstration.

1. Nous désirons montrer que $\sum_{(v_i,v_j)\in C_{df}} p(v_iv_j) < \sum_{(v_i,v_j)\in C} p(v_iv_j)$. Nous savons que

$$\sum_{(v_i,v_j)\in C} p(v_iv_j) = \sum_{(v_i,v_j)\in C_{df}} p(v_iv_j) + \sum_{(v_i,v_j)\in C\setminus C_{df}} p(v_iv_j) \tag{3.10}$$

Or,

$$\begin{aligned}
\Rightarrow \sum_{(v_i,v_j)\in C\setminus C_{df}} p(v_iv_j) &= \sum_{(v_i,v_j)\in P} (1 - x_{v_i} - x_{v_j} + \epsilon) \\
&\quad + (1 - x_{v_f} - x_{v_{f+1}} + \epsilon) + (1 - x_{v_{d-1}} - x_{v_d} + \epsilon) \\
\Rightarrow \quad &= (1 - x_{v_{f+1}} - x_{v_{f+2}} + \epsilon) + (1 - x_{v_{f+2}} - x_{v_{f+3}} + \epsilon) + ... \\
&\quad + (1 - x_{v_{2k}} - x_{v_0} + \epsilon) + (1 - x_{v_0} - x_{v_1} + \epsilon) + ... \\
&\quad + (1 - x_{v_{d-3}} - x_{v_{d-2}} + \epsilon) + (1 - x_{v_{d-2}} - x_{v_{d-1}} + \epsilon) \\
&\quad + (1 - x_{v_f} - x_{v_{f+1}} + \epsilon) + (1 - x_{v_{d-1}} - x_{v_d} + \epsilon)
\end{aligned}$$

$$\Rightarrow \quad = \Big(|P|+1\Big) - 2(x_{v_{f+1}} + x_{v_{f+2}} + ... + x_{v_{2k}} + x_{v_0} + x_{v_1} + ...$$

$$+ x_{v_{d-2}} + x_{v_{d-1}}) + \Big(|P|+1\Big)\epsilon - x_{v_f} - x_{v_d}$$

$$\Rightarrow \quad = |P| - 2\sum_{v_k \in P} x_{v_k} + |P|\epsilon + \Big(1 - x_{v_d} - x_{v_f} + \epsilon\Big)$$

$$\Rightarrow \quad = (1 - x_{v_d} - x_{v_f} + \epsilon) + 2\Big(\frac{|P|}{2} - \sum_{v_k \in P} x_{v_k} + \frac{|P|}{2}\epsilon\Big)$$

Comme $\dfrac{|P|}{2} - \displaystyle\sum_{v_k \in P} x_{v_k} \geq 0$ (voir l'inégalité (2.15) à la page 15), $\epsilon > 0$ et $|P| > 1$, on trouve

$$\frac{|P|}{2} - \sum_{v_k \in P} x_{v_k} + \frac{|P|}{2}\epsilon > 0. \tag{3.11}$$

De plus, $x_{v_d} + x_{v_f} \leq 1$ et $\epsilon > 0 \Longrightarrow 1 - x_{v_d} - x_{v_f} + \epsilon > 0$
D'où

$$\sum_{(v_i,v_j) \in C \setminus C_{df}} p(v_i v_j) > 0. \tag{3.12}$$

Les relations (3.10) et (3.12) nous permettent de conclure que :

$$\sum_{(v_i,v_j) \in C} p(v_i v_j) > \sum_{(v_i,v_j) \in C_{df}} p(v_i v_j) \tag{3.13}$$

2. Supposons que le cycle impair C_{df} contient la source v_0. Ceci implique qu'il existe un plus court chemin de v_0 à $copie(v_0)$ dans le graphe biparti $B(H)$, de la même longueur que le cycle C_{df}. Comme la longueur de C_{df} est inférieure à celle de C (voir le point précédent), alors le chemin qui a fourni le cycle C n'est pas le plus court chemin, ce qui est une contradiction.
3. La démonstration de la troisième observation est triviale puisque, par définition, un trou est un cycle impair sans corde avec au moins 5 sommets.
4. Dans un même cycle impair, il peut exister plusieurs cordes où aucune corde ne domine l'autre (voir la figure 3.7). Chaque corde nous permet d'identifier un trou impair qui possède au moins deux arêtes qui n'appartiennent à aucun autre trou impair. Par conséquent, chaque trou impair fournit une inégalité valide différente des autres inégalités

valides correspondant aux autres trous impairs. □

Les propriétés précédentes nous permettent d'écrire l'algorithme 3.1. Ce dernier effectue le traitement des cordes dans un cycle impair donné.

Exemple 3.3.5 *Dans l'exemple de la figure 3.9, nous avons un cycle impair* $C = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$. *Le cycle* C *contient cinq cordes* $\{v_1, v_5\}$, $\{v_1, v_7\}$, $\{v_2, v_8\}$, $\{v_3, v_7\}$ *et* $\{v_4, v_6\}$. *En appliquant la définition 3.3.3, nous constatons que la corde* $\{v_1, v_5\}$ *domine la corde* $\{v_1, v_7\}$ *et la corde* $\{v_4, v_6\}$ *domine les cordes* $\{v_3, v_7\}$, $\{v_2, v_8\}$. *Le cycle* C_{15} *composé des sommets* $\{v_1, v_2, v_3, v_4, v_5\}$ *est donc un trou impair, de même que le triangle* C_{46} *composé des sommets* $\{v_4, v_5, v_6\}$.

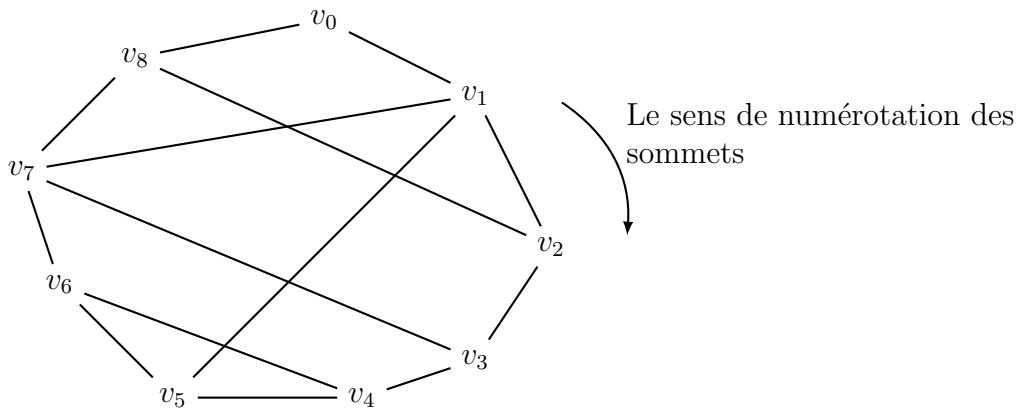

Figure 3.9 Exemple de cycle avec plusieurs cordes

3.3.3 Les inégalités valides

La procédure 3.1 nous permet d'identifier des trous impairs dans le graphe de conflit $L(H)$. Par la proposition 3.3.6, nous démontrons que chaque trou impair dans le graphe de conflit $L(H)$ correpond à un sous-multigraphe conflictuel épineux dans le sous-multigraphe partiel H.

Proposition 3.3.6 *Tout trou impair* $C = \{v_0, v_1, ..., v_{|C|-1}\}$ *de* $L(H)$ *correspond à un sous-multigraphe conflictuel épineux* $G' = (S_1 \cup S_2 \cup S_3, F)$ *dans* H *avec* $|C| = |S_1| + 2|S_2|$ *et* $\sum_{\ell \in C} x_\ell = \sum_{(i,j,k) \in F} X_{ij}^k$.

Algorithme 3.1 Traitement de Cordes($C, ListeCycles$)

Soit $C' = \{a_0, a_1, ..., a_{2k}\}$ le cycle conflictuel de H correspondant au cycle impair $C = \{v_0, v_1, ..., v_{2k}\}$ de $L(H)$;
Soit la liste des cordes $ListeCordes = \emptyset$;
Soit la liste des cycles impairs $ListeCycles$;
pour $i := 1$ **à** $2k - 1$ **faire**
  si *i est pair* **alors**
    $\ell := 2k$;
  sinon
    $\ell := 2k - 1$;
  finsi
  pour $j = i + 2$ **à** ℓ **pas** 2 **faire**
    si les arcs a_i et a_j sont en conflit **alors**
      ajouter la corde (v_i, v_j) à la liste des cordes $ListeCordes$;
    finsi
  finpour
finpour
si *ListeCordes est vide* **alors**
  ajouter C' à la liste des trous impairs $ListeCycles$;
sinon
  pour *chaque* $CORDE \in ListeCorde$ **faire**
    si $CORDE$ *n'est dominée par aucune autre corde de ListeCordes* **alors**
      ajouter le trou impair engendré par $CORDE$ à $ListeCycles$;
    finsi
  finpour
finsi

Démonstration. Cette proposition provient de certaines définitions précédentes et est facile à vérifier. Soit $C = \{v_0, v_1, ..., v_{|C|-1}\}$ un trou impair dans $L(H)$ de longueur $\sum_{\ell \in C} x_\ell$. Soit v_l et v_{l+1} deux sommets adjacents de C correspondant respectivement aux arcs (i, j, k) et (i', j', k') dans H. Les acrs (i, j, k) et (i', j', k') sont en conflit car les sommets correspondants sont en conflit. Selon la définition 2.1.2, (i, j, k) et (i', j', k') sont aussi adjacents. En appliquant le même raisonnement à tous les sommets de C deux à deux adjacents, nous obtenons un sous-multigraphe conflictuel épineux $G' = (S, F)$ de H. Comme C est un trou impair, alors G' est un cycle impair. De plus, le nombre de sommets de C est égal au nombre d'arêtes de G'. Vu que C est un trou impair, alors on sait qu'au niveau de chaque sommet de G' il existe exactement deux arcs en conflit. Par conséquent, pour chaque sommet intermédiaire de G', il existe une et une seule épine. Nous pouvons donc conclure que $|C| = |F| = |S_1| + |S_2| + |S_3| = |S_1| + 2|S_2|$ et $\sum_{\ell \in C} x_\ell = \sum_{(i,j,k) \in F} X_{ij}^k$ □

3.3.4 Enrichissement des inégalités valides

Afin d'enrichir les inégalités valides, nous appliquons une procédure de liftage avec des arcs de flux non nul ensuite avec des arcs de flux nul. Soit $G' = (S, F)$ un sous-multigraphe conflictuel épineux, où $F = \{a_0 = a_{2k+1}, a_1, a_2, ..., a_{2k}\}$ représente l'ensemble des arcs. Rappelons que pour $i = 0, 1, ..., 2k$, les arcs a_i et a_{i+1} sont en conflit.

Définition 3.3.7 *Un ensemble Q d'arcs du sous-multigraphe partiel H est une clique de conflit si tous les arcs de Q sont deux à deux en conflit.*

À partir des arcs du sous-multigraphe conflictuel épineux $G' = (S, F)$, nous construisons un ensemble $\mathbf{Q}$ des cliques $\{Q_1, Q_2, ..., Q_{2k+1}\}$, où $Q_i = \{a_{i-1}, a_i\}$, pour $i = 1, 2, ..., 2k + 1$. Initialement chaque clique de conflit contient deux arcs qui sont adjacents et en conflit.

Comme deux arcs en conflit ont au moins une extrémité en commun, désignons par $S(Q_i)$ l'ensemble des extrémités communes à tous les arcs de Q_i.

$$S(Q_i) = \begin{cases} \{s\} & \text{si } a_{i-1} \text{ et } a_i \text{ ne sont pas des arcs parallèles et} \\ & s \text{ est l'extrémité commune de } a_{i-1} \text{ et } a_i \\ \{u, v\} & \text{si } a_{i-1} \text{ et } a_i \text{ sont des arcs parallèles où } u \\ & \text{et } v \text{ sont leurs extrémités communes} \end{cases}$$

Notons que deux cliques distinctes peuvent avoir le même ensemble d'extrémités.

Soit A' l'ensemble des arcs de H. Pour un arc $a = (u, v, k) \in A'$, posons $K(a) = \{Q \in \mathbf{Q} : S(Q) \cap \{u, v\} \neq \emptyset\}$ l'ensemble des cliques contenant l'arc a .

Afin de renforcer l'inégalité valide trouvée, nous construisons l'ensemble des cliques $\mathbf{Q}$ du sous-multigraphe partiel la représentant. Ensuite, s'il existe un arc $(i, j, k) \in A'$ et $(i, j, k) \notin F$ tel que en ajoutant cet arc à une clique de $\mathbf{Q}$, cette dernière préserve ses propriétés, alors nous ajoutons l'arc (i, j, k) à F. L'algorithme 3.2 effectue l'enrichissement des inégalités valides avec des arcs de flux non nul (strictement positif). Ensuite, nous effectuons l'enrichissement avec des arcs de flux nul. Dans ce cas, l'algorithme 3.2 reste valable et le seul changement se fera au niveau de la ligne 5, où nous remplaçons la condition $X^k_{uv} > 0$ par $X^k_{uv} = 0$. Le nouveau algorithme est nommé LiftNul(A', F).

Exemple 3.3.8 *Dans l'exemple représenté à la figure 3.10, les arcs a_i sont les arcs du cycle conflictuel épineux avant l'application de l'algorithme 3.2 et les arcs b_j sont les arcs ajoutés par la procédure de liftage.*

Étape initiale

$C' = \{a_0 = a_{15}, a_1, ..., a_{14}\}$ est le cycle conflictuel épineux. L'ensemble des cliques est $\boldsymbol{Q} = \{Q_1, Q_2, ..., Q_{15}\}$, où

Algorithme 3.2 LiftPos(A', F)

1: //A' est l'ensemble des arcs du multigraphe partiel H et F est l'ensemble des arcs du sous-multigraphe conflictuel épineux
2: Former les cliques Q_i, $i = 1, 2, ..., 2k + 1$ et définir pour chaque clique l'ensemble $S(Q_i)$
3: **pour** chaque arc $a = (u, v, k) \in A'$ **faire**
4: $K(a) := \emptyset$
5: **si** $X^k_{uv} > 0$ et ni l'origine ni la destination de a ne sont des dépôts **alors**
6: **pour** chaque $Q_i \in$ **Q faire**
7: **si** $S(Q_i) \cap \{u, v\} \neq \emptyset$ **alors**
8: ajouter Q_i à $K(a)$
9: **finsi**
10: **finpour**
11: **si** il existe deux cliques distinctes Q_i et $Q_j \in K(a)$ **telles que** $Q_i \cup \{a\}$ *et* $Q_j \cup \{a\}$ sont des cliques **alors**
12: $Q_i := Q_i \cup \{a\}$
13: $Q_j := Q_j \cup \{a\}$
14: $F := F \cup \{a\}$
15: **finsi**
16: **finsi**
17: **finpour**

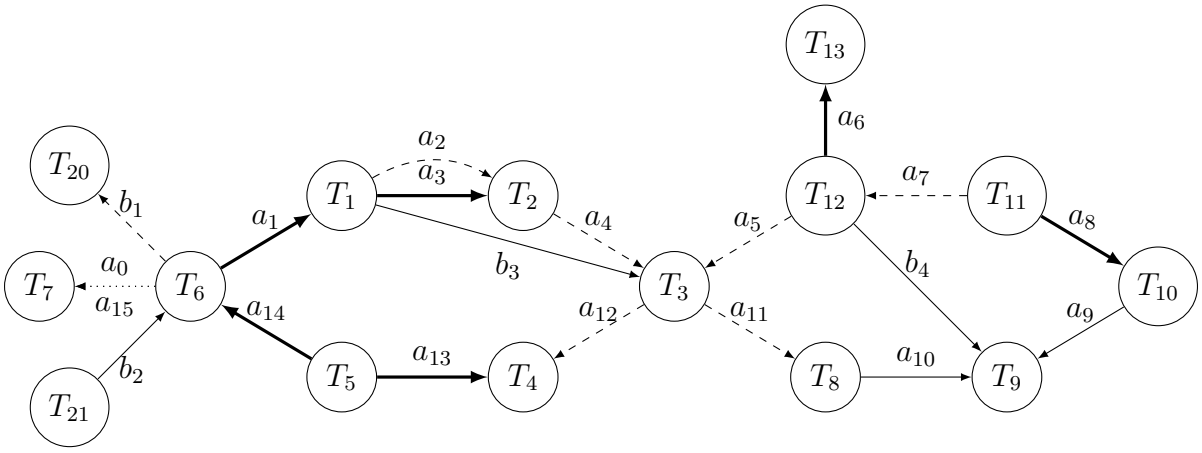

Figure 3.10 Exemple de liftage d'un cycle conflictuel épineux avec un point d'articulation

$Q_1 = \{a_0, a_1\}$ *et* $S(Q_1) = \{T_6\}$,
$Q_2 = \{a_1, a_2\}$ *et* $S(Q_2) = \{T_1\}$,
$Q_3 = \{a_2, a_3\}$ *et* $S(Q_3) = \{T_1, T_2\}$,
...
$Q_{15} = \{a_{14}, a_0\}$ *et* $S(Q_{15}) = \{T_6\}$.

Étape intermédiaire

Soit b_1 un arc de A' ($b_1 \notin F$) et ni son origine (le sommet T_6) ni sa destination (le sommet T_{20}) ne sont des dépôts. On remarque que $S(Q_1) \cap \{T_6, T_{20}\} = \{T_6\}$ et $S(Q_{15}) \cap \{T_6, T_{20}\} = \{T_6\}$, donc $K(b_1) = \{Q_1, Q_{15}\}$.
Puisque $Q_1 \cup \{b_1\}$ et $Q_{15} \cup \{b_1\}$ sont des cliques de conflit, alors $Q_1 = \{a_0, a_1, b_1\}$, $Q_{15} = \{a_{14}, a_0, b_1\}$ et $F = \{a_0 = a_{15}, a_1, ..., a_{14}, b_1\}$
Cette étape s'applique pour chaque arc de A' possédant au moins une extémité en commun avec au moins deux cliques distinctes.

Étape finale

La procédure de liftage se termine quand tous les arcs de A' sont examinés. Pour cet exemple, à la fin de la procédure les cliques modifiées sont
$Q_1 = \{a_0, a_1, b_1, b_2\}$,
$Q_{15} = \{a_{14}, a_0, b_1, b_2\}$,
$Q_2 = \{a_1, a_2, b_3\}$,
$Q_5 = \{a_4, a_5, b_3\}$,
$Q_6 = \{a_5, a_6, b_4\}$,
$Q_{10} = \{a_9, a_{10}, b_4\}$,
et le cycle conflictuel épineux représenté à la figure 3.10 devient $\{a_0 = a_{15}, a_1, ..., a_{14}, b_1, b_2, b_3, b_4\}$

Remarque 3.3.9 *Le sous-multigraphe $G'' = (S, F'')$ obtenu en liftant le sous-multigraphe conflictuel épineux $G' = (S, F)$ est aussi un sous-multigraphe conflictuel épineux de G.*

Proposition 3.3.10 *Soit $G'' = (S, F'')$ le sous-multigraphe conflictuel épineux obtenu en liftant le sous-multigraphe conflictuel épineux $G' = (S, F)$, L'inégalité*

$$\sum_{(i,j,k)\in F''} X_{ij}^k \leq \left\lfloor \frac{|S_1|}{2} \right\rfloor + |S_2| \tag{3.14}$$

est valide pour $(MDVSP)$.

Démonstration.

En liftant un sous-multigraphe épineux G', nous gardons les mêmes sommets et chaque arc ajouté est en conflit avec tous les arcs qui lui sont adjacents, nous avons donc $|S| = |S_1|+2|S_2|$ sommets dans le sous-multigraphe G''. De plus, au niveau de chaque sommet $s \in S$, $\sum_{(s,j,k)\in F''} X_{sj}^k + \sum_{(i,s,k)\in F''} X_{is}^k \leq 1$ est valide pour $(MDVSP)$. La somme de toutes les inégalités valides obtenues au niveau de chaque sommet du sous-multigraphe conflictuel épineux G'' est :

$$\sum_{s=1}^{|S|}\left(\sum_{(s,j,k)\in F''} X_{sj}^{k}+\sum_{(i,s,k)\in F''} X_{is}^{k}\right)\leq |S| \tag{3.15}$$

Comme chaque arc possède deux extrémités, X_{ij}^{k} apparaît deux fois dans la somme précédente, d'où :

$$\sum_{(i,j,k)\in F''} 2X_{ij}^{k}\leq |S|=|S_1|+2|S_2| \tag{3.16}$$

$$\tag{3.17}$$

Par conséquent,

$$\sum_{(i,j,k)\in F''} X_{ij}^{k}\leq \left\lfloor \frac{|S_1|}{2}\right\rfloor +|S_2|. \tag{3.18}$$

est valide pour $(MDVSP)$. □

3.3.5 Algorithme de séparation des inégalités de cycle impair conflictuel

En utilisant la solution fractionnaire du MDVSP, nous construisons les graphes $H=(V',A')$, $L(H)=(A',E)$ et $B(H)=(A',A'',E')$. Ensuite, pour chaque sommet de $i\in A'$, nous cherchons le plus court chemin du sommet i à $copie(i)\in A''$, afin d'identifier des cycles impairs, nous appliquons le traitement des cordes pour séparer les trous impairs, s'ils existent. Nous construisons le sous-multigraphe conflictuel épineux correspondant. Nous enrichissons chaque sous-multigraphe conflictuel épineux trouvé en liftant avec des arcs de flux non nul ; ensuite avec des arcs de flux nul. Rappelons qu'un sous-multigraphe conflictuel épineux lifté correspond à une inégalité valide. Après la séparation des inégalités valides, nous les ajoutons au problème à résoudre. Étant donné qu'il peut exister un grand nombre d'inégalités valides, il est préférable de n'ajouter que les plus fortes (en terme de violation).

Il est possible d'identifier toutes les inégalités valides, ensuite de vérifier si la valeur de la violation est supérieure à $SeuilVio$; dans ce cas, l'inégalité valide est ajoutée au problème. Mais nous pouvons améliorer le temps d'exécution en supprimant certains sommets et arcs du graphe biparti $B(H)$ avant même de faire le traitement des cordes. Nous avons remarqué qu'il existe une relation entre le poids du plus court chemin et le seuil de violation, et par conséquent, entre ce dernier et le poids des arcs de $B(H)$.

Lemme 3.3.11 *Soit $G' = (S, F)$ un sous-multigraphe épineux du graphe $H = (A', V')$ et $SeuilVio$ un seuil minimal de violation fixé. La relation $\sum_{(i,j,k)\in F} X_{ij}^k - \left\lfloor \frac{|S_1|}{2} \right\rfloor - |S_2| \geq SeuilVio$ est vérifiée si et seulement si le poids du plus court chemin de v_0 à v_0' dans $B(H)$ est inférieur ou égal à $1 - 2SeuilVio$.*

Démonstration.

Nous désirons que la violation soit supérieure ou égale à $SeuilVio$. Nous avons donc

$$\sum_{(i,j,k)\in F} X_{ij}^k - \left\lfloor \frac{|S_1|}{2} \right\rfloor - |S_2| \geq SeuilVio \tag{3.19}$$

$$\iff 2 \sum_{(i,j,k)\in F} X_{ij}^k - (|S_1| + 2|S_2| - 1) \geq 2SeuilVio \tag{3.20}$$

$$\iff 2 \sum_{(i,j,k)\in F} X_{ij}^k - (|S_1| + 2|S_2|) \geq 2SeuilVio - 1 \tag{3.21}$$

$$\iff |S_1| + 2|S_2| - 2 \sum_{(i,j,k)\in F} X_{ij}^k \leq 1 - 2SeuilVio. \tag{3.22}$$

Nous avons déjà montré qu'à tout trou impair C de $L(H)$ correspond un sous-multigraphe conflictuel épineux $G' = (S, F)$ de H, tel que $|C| = |F| = |S_1| + 2|S_2| = 2k + 1$ et $\sum_{v_\ell \in C} x_{v_\ell} = \sum_{(i,j,k)\in F} X_{ij}^k$ (voir la proposition 3.3.6). Par conséquent, la relation (3.22) devient

$$|C| - 2 \sum_{v_\ell \in C} x_{v_\ell} \leq 1 - 2SeuilVio \tag{3.23}$$

$$\iff 2k + 1 - 2 \sum_{v_\ell \in C} x_{v_\ell} \leq 1 - 2SeuilVio. \tag{3.24}$$

L'expression $\left(2k+1-2 \sum_{v_\ell \in C} x_{v_\ell}\right)$ représente le poids du plus court chemin de v_0 à v_0' (voir la relation (2.13)) dans la graphe biparti $B(H)$ et l'expression est aussi égale à la somme des poids des arêtes du trou impair C dans $L(H)$. □

Lemme 3.3.12 *Si le poids du plus court chemin de v_0 à v_0' dans le graphe biparti $B(H)$ est inférieur ou égal à $1 - 2SeuilVio$, alors le poids de chaque arc du plus court chemin de v_0 à v_0' est inférieur ou égal à $1 - 2SeuilVio$.*

Démonstration.

Le poids du trou impair est égal à la somme des poids des arêtes appartenant à ce trou. Puisque le poids du trou impair est inférieur à $1-2SeuilVio$ (ce qui est prouvé dans le lemme 3.3.11), alors le poids de chaque arête est aussi inférieur à $1-2SeuilVio$ (car le poids de chaque arête (u,v) dans $L(H)$ est non négatif puisqu'il est égal à $1-x_u-x_v \geq 0$). □

Par conséquent, tous les arcs dont le poids est supérieur à $1-2SeuilVio$ sont inutiles car si au moins un arc de poids supérieur à $1-2SeuilVio$ appartient au plus court chemin de v_0 à v_0' dans le graphe $B(H)$, alors le poids de ce chemin sera supérieur à $1-2SeuilVio$. Dans ce cas, il est préférable de les éliminer ; ceci nous permettra de diminuer la dimension du graphe $L(H)$, et par conséquent, celle du graphe $B(H)$. Pour nos expériences numériques , nous avons fixé $SeuilVio$ à 0.2. Pour réduire encore plus la taille du graphe $B(H)$, nous avons redéfini l'ensemble des arcs de sous-multigraphe H tel que $A' = \{(i,j,k) \in A \mid \frac{1}{8} < X_{ij}^k < 1\}$.

La procédure de séparation des inégalités de cycle impair conflictuel est donnée par l'algorithme 3.3. Au pire des cas, la complexité de cet algorithme est égale à $|A'|$ multiplié par la complexité de l'algorithme de Dijkstra. Nous avons implémenté l'algorithme de Dijkstra en utilisant les tas binaires, ce qui nous a permet d'avoir une complexité de $O(|A'|log(|V'|))$. Par conséquent, la complexité de l'algorithme 3.3 est $O(|A'|^2log(|V'|))$.

3.4 Séparation des inégalités de clique de cardinalité 3

Une clique de cardinalité 3 est un sous-multigraphe conflictuel épineux $G' = (S,F)$, tel que :

- $|S| = |S_1| = 3$,
- tous les arcs adjacents dans F sont en conflit.

Normalement la méthode de séparation précédemment présentée permet de déterminer les inégalités de clique de cardinalité 3. Mais, afin d'améliorer le temps de résolution de la méthode de séparation des inégalités de cycle impair conflictuel, nous avons imposé que la violation des inégalités valides doit être supérieure ou égale à un seuil de violation, une valeur donnée. Étant donné que les inégalités de clique sont importantes, nous avons donc utilisé la procédure décrite par l'algorithme 3.4, afin de séparer toutes les inégalités de clique de cardinalité 3 d'une violation strictement positive. Au pire des cas, la compexité de l'algorithme 3.4 est $O(|V||A|)$.

3.5 Séparation des inégalités de coupe impaire

La deuxième méthode de détermination des inégalités valides pour le MDVSP que nous proposons consiste à utiliser le sous-multigraphe partiel orienté, $H = (V', A')$ pour construire le sous-multigraphe des chemins réguliers $G^* = (V', \mathcal{P})$, où $\mathcal{P}$ repésente l'ensemble des che-

Algorithme 3.3 Séparation d'inégalités de Cycle Impair($Solution, ListInégValid$)

extraire la solution de la relaxation linéaire
fixer le paramètre $SeuilVio$
créer les graphes $H = (V', A')$, $L(H) = (A', E)$ et $B(H) = (A', A'', E')$
$ListeCycles = \emptyset$, $ListeConf = \emptyset$, $ListeConfLift = \emptyset$,
pour chaque sommet $i \in A'$ et n'appartenant pas à un cycle C de $ListeCycles$ **faire**
  Dijkstra($B(H), w, i$)
  si il existe un plus court chemin de i à $copie(i)$ de poids inférieur à $1 - 2SeuilVio$ **alors**
    $C :=$ le cycle impair dans $L(H)$ correspondant au plus court chemin de i à $copie(i)$ dans $B(H)$
    TraitementCordes($C, ListeCycles$)
  finsi
finpour
pour chaque trou impair $C \in ListeCycles$ **faire**
  identifer le sous-multigraphe conflictuel épineux $G' = (S, F)$ correspondant à C
  ajouter $G' = (S, F)$ à $ListeConf$
finpour
pour chaque sous-multigraphe conflictuel épineux $G' = (S, F) \in ListeConf$ **faire**
  LiftPos(A',F)
  si G' n'existe pas dans $ListeConfLift$ (après liftage) **alors**
    ajouter G' à $ListeConfLift$
  finsi
finpour
pour chaque sous-multigraphe conflictuel épineux $G' \in ListeConfLift$ **faire**
  LiftNul(A',F)
  ajouter l'inégalité valide relative à G' à $ListInégValid$
finpour

mins réguliers (voir 3.5.1). Ensuite, nous formulons un problème auxiliaire en nombres entiers qui nous permet d'identifier des coupes impaires de capacité strictement inférieure à 1. Nous montrons que chaque coupe impaire correspond à un sous-multigraphe conflictuel non épineux. Après l'ajout des épines au niveau de chaque noeud intermédiaire, la coupe impaire correspond à un sous-multigraphe conflictuel épineux dans H. Nous montrons que les sous-multigraphes conflictuels épineux nous fournissent des inégalités valides. Nous utilisons une procédure de liftage pour enrichir les inéglités valides. Dans cette section nous présentons les détails de chaque étape.

3.5.1 Création du graphe des chemins

Avant de présenter la méthode de construction du sous-multigraphe des chemins, nous présentons la définition d'un chemin régulier.

Algorithme 3.4 Séparation d'inégalités de Clique de cardinalité 3($Solution, ListInégValid$)

```
1: Soit G = (V, A),
2: pour tout sommet i ∈ V faire
3:   pour toute paire d'arcs (i, j, k) ∈ δ+(i) et (i, j', k') ∈ δ+(i) faire
4:     Cycle = faux,
5:     si j ≠ j' et X^k_ij > 0 et X^k'_ij' > 0 alors
6:       pour chaque arc (j, j', k'') ∈ δ+(j) faire
7:         si k ≠ k'' alors
8:           Cycle = vrai,
9:           Soit S = {i, j, j'},
10:          Soit F = {(i, j, k), (i, j', k'), (j, j', k'')},
11:          construire G' = (S, F) un sous-multigraphe partiel de G,
12:          lifter G' = (S, F),
13:          ajouter l'inégalité valide relative à G' à ListInégValid
14:        finsi
15:      finpour
16:      si Cycle = faux alors
17:        pour chaque arc (j', j, k'') ∈ δ+(j') faire
18:          si k' ≠ k'' alors
19:            Cycle = vrai,
20:            Soit S = {i, j, j'},
21:            Soit F = {(i, j, k), (i, j', k'), (j', j, k'')},
22:            construire G' = (S, F) un sous-multigraphe partiel de G,
23:            lifter G' = (S, F),
24:            ajouter l'inégalité valide relative à G' à ListInégValid
25:          finsi
26:        finpour
27:      finsi
28:    finsi
29:  finpour
30: finpour
```

Définition 3.5.1 *Dans un sous-multigraphe partiel orienté, un chemin* P_{i_1,i_h} =(T_{i_1}, T_{i_2},..., T_{i_h}) *est dit régulier si*

- *le chemin* P_{i_1,i_h} *mène d'un noeud de conflit à un noeud de conflit (c'est-à-dire d'une source à un puits, d'une source à un noeud ordinaire, d'un noeud ordinaire à un noeud ordinaire, ou d'un noeud ordinaire à un puits),*
- $X^k_{i_l,i_{l+1}} > 0$ *pour tout arc* (i_l, i_{l+1}, k), $l = 1, ..., h-1$ *appartenant au chemin* P_{i_1,i_h},
- *tous les arcs du chemin* P_{i_1,i_h} *sont de même couleur,*
- *le poids du chemin* P_{i_1,i_h} *est égal à la valeur minimale de* $X^k_{i_l,i_{l+1}}$ *pour tout arc* (i_l, i_{l+1}, k), $l = 1, ..., h-1$, *appartenant à* P_{i_1,i_h}.

Remarque 3.5.2 *Notons qu'un noeud reliant deux arcs appartenant au même chemin régulier est un noeud intermédiaire.*

L'ensemble des sommets du sous-multigraphe partiel $H = (V', A')$ est partitionné en trois sous-ensembles : V'_s est le sous-ensemble des sommets source, V'_p est le sous-ensemble des sommets puits et V'_a est le sous-ensemble des sommets qui ne sont ni source ni puits.

À partir d'un sommet $s \in V'_s$, et tant qu'il existe un arc sortant de s, nous déterminons un chemin de couleur k reliant le sommet s à un autre sommet $t \in V' \backslash V'_s$, tel qu'il n'existe aucun arc sortant du sommet t de couleur k. Le poids du chemin P_{st} est égal à $w_{st} = \min\limits_{(i,j,k)\in P_{st}} \{(X^k_{ij})\}$ et pour chaque arc $(i, j, k) \in P_{st}$, nous posons $X^k_{ij} = X^k_{ij} - w_{st}$ et si $X^k_{ij} = 0$, alors nous supprimons l'arc (i, j, k) du sous-multigraphe partiel H. Nous répétons la même opération pour chaque sommet appartenant à V'_s, ensuite pour chaque sommet appartenant à V'_a. Soit $\mathcal{P}$ l'ensemble de tous les chemins réguliers $P_{i,j}$ reliant le noeud $i \in V'$ au noeud $j \in V'$.

Soit $G^* = (V', \mathcal{P})$ un sous-multigraphe partiel, où V' représente l'ensemble des sommets et $\mathcal{P}$ l'ensemble des chemins réguliers. Soit $A = (a_{ip})_{i\in V', p\in\mathcal{P}}$ la matrice d'incidence sommets-chemins, correspondant à la solution courante de la relaxation linéaire. Le coefficient a_{ip} prenda la valeur 1 si le sommet i appartient au chemin p et 0 sinon.

3.5.2 Formulation du problème auxiliaire

Notre objectif est de déterminer un ensemble de sommets $S_1 \subset V'$ de cardinalité impaire, tel que l'intersection de tout chemin $p \in \mathcal{P}$ avec S_1 soit de cardinalité au plus 2, et que le poids total des chemins ayant une intersection avec S_1 de cardinalité 1 soit minimal.

Considérons le sous-multigraphe $G^* = (V', \mathcal{P})$, nous associons à chaque sommet $i \in V'$ une variable binaire x_i, qui prend la valeur 1 si le sommet i appartient à l'ensemble S_1, et 0 sinon. À chaque chemin $p \in \mathcal{P}$, nous associons deux variables binaires y_p et z_p. La variable y_p prend la valeur 0 si l'intersection du chemin p avec l'ensemble S_1 est vide, et 1 sinon. Par contre, la variable z_p prend la valeur 1 si l'intersection du chemin p avec l'ensemble S_1 est de cardinalité 1, et 0 sinon.

Le problème de séparation des coupes impaires de capacité minimale se formule comme un programme linéaire en nombres entiers.

$$(CoupeImp)\begin{cases} \min \quad \sum_{p\in\mathcal{P}} w_p z_p & & (3.25)\\ \text{s.c.} \quad \sum_{i\in V'} a_{ip}x_i + z_p - 2y_p = 0 & \forall p\in\mathcal{P} & (3.26)\\ \sum_{i\in V'} x_i = 2k+1 & & (3.27)\\ k_{min} \leqslant k \leqslant k_{max} & & (3.28)\\ k \geqslant 0, \quad k\in\mathbb{Z} & & (3.29)\\ x_i \in \{0,1\}, & \forall i\in V' & (3.30)\\ z_p, y_p \in \{0,1\}. & \forall p\in\mathcal{P} & (3.31) \end{cases}$$

L'objectif (3.25) minimise le poids total des chemins ayant une intersection avec S_1 de cardinalité 1. Les contraintes (3.26) garantissent que chaque chemin contient au plus deux sommets appartenant à S_1. La contrainte (3.27) assure que la cardinalité de S_1 soit impaire. Les contraintes (3.28) et (3.29) définissent le domaine de la variable k. Notons que nous limitons la taille de l'ensemble S_1 dans le but d'accélérer la résolution du problème. Les contraintes (3.30) et (3.31) indiquent que les variables x_i, y_p et z_p sont binaires.

Le problème $(CoupeImp)$ est un programme linéaire en nombres entiers que nous résolvons à l'aide de la méthode de séparation et évaluation progressive.

Proposition 3.5.3 *Toute solution réalisable du problème* $(CoupeImp)$ *correspond à une coupe impaire de capacité minimale.*

Démonstration.

Le problème de coupe impaire de capacité minimale consiste à déterminer, dans un graphe donné, un sous-ensemble de sommets U de cardinalité impaire, tel que le poids total des arêtes appartenant à $\delta(U)$ est minimal.

Soit $(S_1, \mathcal{P}_1, \mathcal{P}_2)$ une solution réalisable du problème $(CoupeImp)$, $S_1 = \{i \in V' |\ x_i = 1\}$ est l'ensemble des sommets. $\mathcal{P}_1 = \{p \in \mathcal{P} |\ y_p = 1 \text{ et } z_p = 1\}$ représente l'ensemble des chemins p tels que l'intersection de p avec S_1 est de cardinalité 1, $\mathcal{P}_2 = \{p \in \mathcal{P} |\ y_p = 1 \text{ et } z_p = 0\}$ représente l'ensemble des chemins p tels que l'intersection de p avec S_1 est de cardinalité 2. La valeur de la solution est $W^* = \sum_{p\in\mathcal{P}} w_p z_p$.

Donc, $\sum_{i\in S_1} x_i = 2k+1$, ce qui implique que l'ensemble S_1 est de cardinalité impaire. La valeur de la solution est $W^* = \sum_{p\in\mathcal{P}_1} w_p$, en d'autres termes, l'objectif minimise le poids

total des chemins appartenant à $\delta(S_1)$. Par conséquent, la solution réalisable $(S_1, \mathcal{P}_1, \mathcal{P}_2)$ correspond à une coupe impaire de capacité minimale définie par $(S_1, \mathcal{P}_1, \mathcal{P}_2)$. □

3.5.3 Les inégalités valides

La résolution du problème auxiliaire ($CoupeImp$) nous permet d'identifier une coupe impaire. En considérant une solution réalisable $(S_1, \mathcal{P}_1, \mathcal{P}_2)$ du problème ($CoupeImp$), nous construissons un sous-multigraphe partiel $G' = (S_1 \cup S_2, F)$, en suivant les étapes suivantes :

- Pour tout chemin $p \in \mathcal{P}_2$,
  - ✓ extraire le sous-chemin régulier $p' = \{i_k, i_{k+1}, ..., i_{l-1}, i_l\}$ tel que p' relie les sommets i_k et i_l de S_1, c'est-à-dire $x_{i_k} = x_{i_l} = 1$ et $x_{i_{k+1}} = x_{i_{k+2}} = ... = x_{i_{l-1}} = 0$.
  - ✓ extraire l'ensemble des sommets intermédiaires du sous-chemin régulier p', soit $I_{p'} = \{i_{k+1}, i_{k+2}, ..., i_{l-1}\}$.
  - ✓ extraire l'ensemble des arcs reliant toute paire de sommets du sous-chemin régulier p', soit $F'_{p'} = \{(i, j, c) \in A' \mid i, j \in p'\}$.
- extraire l'ensemble de tous les sous-chemins réguliers p', soit $\mathcal{P}'_2 = \bigcup p'$.
- extraire l'ensemble de tous les sommets intermédiaires, soit $S_2 = \bigcup_{p' \in \mathcal{P}'_2} I_{p'}$.
- extraire l'ensemble de tous les arcs des sous-chemins réguliers $p' \in \mathcal{P}'_2$, soit $F = \bigcup_{p' \in \mathcal{P}'_2} F'_{p'}$.

Nous avons construit un sous-multigraphe partiel $G' = (S_1 \cup S_2, F)$, où S_1 est l'ensemble des sommets de conflit, S_2 est l'ensemble des sommets intermédiaires, et F représente l'ensemble des arcs induits par l'ensemble des sommets $S_1 \cup S_2$.

Proposition 3.5.4 *Soit $G' = (S, F)$ un sous-multigraphe partiel de $G^* = (V', \mathcal{P}'_2)$, où $S = S_1 \cup S_2$ et $F = \bigcup_{p' \in \mathcal{P}'_2} F'_{p'}$. G' est un sous-multigraphe conflictuel non épineux.*

Démonstration.

Le sous-multigraphe partiel $G' = (S, F)$ est conflictuel non épineux si les conditions suivantes sont vérifiées :

- $S_1 \cup S_2$ ne contient aucun dépot,
- $|S_1|$ est impaire et au moins égale à 3,
- S_2 est l'ensemble des noeuds intermédiaires,
- S_1 est l'ensemble des noeuds de conflit.

✓ La première condition est vérifiée, puisque nous savons que $S_1 \cup S_2 \subseteq V'$ et V' ne contient aucun dépôt.

✓ La deuxième condition est vérifiée, mais il faut préciser que même si $|S_1|$ est paire et que les autres conditions sont vérifiées la proposition reste valide.

✓ Selon la définition 3.2.1, un noeud $i \in S$ est dit intermédiaire, si $\delta_F^-(i) \neq \emptyset$, $\delta_F^+(i) \neq \emptyset$ et $Col(\delta_F^-(i)) \cap Col(\delta_F^+(i)) \neq \emptyset$. Par construction, tout sommet $i \in S_2$ est un sommet intermédiaire d'un sous-chemin régulier. En d'autres termes, il existe au moins un arc entrant et un arc sortant, et pour chaque arc entrant de couleur k, il existe un arc sortant de même couleur. Donc, tout sommet $i \in S_2$ vérifie les conditions de la définition 3.2.1. Par conséquent, S_2 est l'ensemble des noeuds intermédiaires.

✓ Selon la définition 3.2.1, un noeud $i \in S$ est dit noeud de conflit, s'il est source, puits, ou noeud ordinaire. Par construction, S_1 est l'ensemble de tous les sources et puits des sous-chemins réguliers, pour tout noeud $i \in S_1$, il existe trois cas possibles :

cas 1 : $\delta_F^-(i) = \emptyset$, donc i est un noeud source,

cas 2 : $\delta_F^+(i) = \emptyset$, donc i est un noeud puits,

cas 3 : $\delta_F^-(i) \neq \emptyset$ et $\delta_F^+(i) \neq \emptyset$, c'est-à-dire que i est puits d'un sous-chemin régulier p'_1 et source d'un autre sous-chemin régulier p'_2. Si $Col(\delta_F^-(i)) \cap Col(\delta_F^+(i)) = \emptyset$, alors i est noeud ordinaire, sinon i est un noeud intermédiaire, donc il faut poser $S_1 = S_1 \backslash \{i\}$ et $S_2 = S_2 \cup \{i\}$.

Par conséquent, tous les sommets de S_1 sont des noeuds de conflit. □

Exemple 3.5.5 *Dans la figure 3.11, nous avons un exemple de coupe impaire qui nous permet d'identifier un sous-multigraphe non épineux, où* $S_1 = \{T_1, T_4, T_5\}$ *est l'ensemble des noeuds de conflit,* $S_2 = \{T_2, T_3, T_6\}$ *est l'ensemble des noeuds intermédiaires. Il existe trois sous-chemins réguliers :* $p'_1 = \{T_1, T_2, T_3, T_5\}$, $p'_2 = \{T_4, T_5\}$ *et* $p'_3 = \{T_4, T_6, T_1\}$.

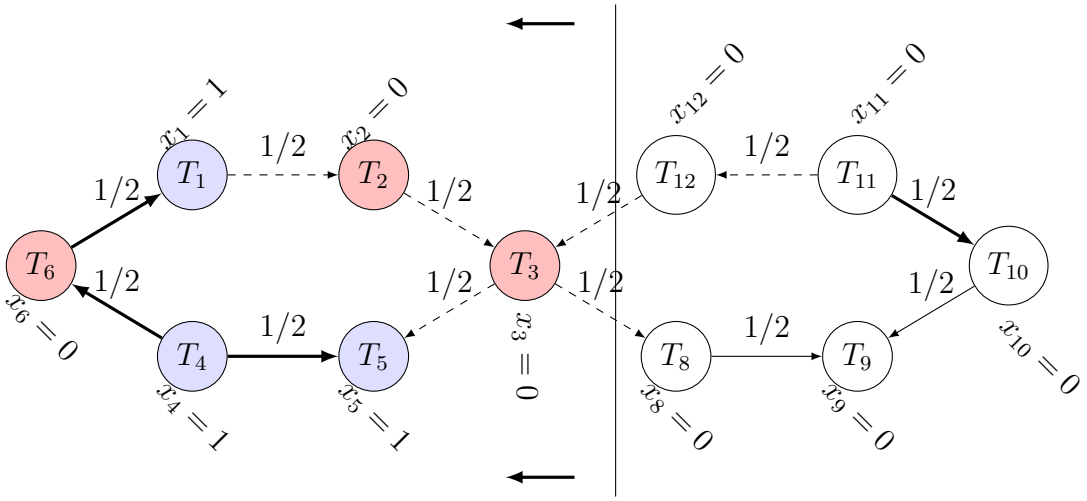

Figure 3.11 Exemple de coupe impaire de cardinalité 3 et de capacité égale à 0.0

3.5.4 Enrichissement et ajout des épines

Afin d'enrichir le sous-multigraphe conflictuel non épineux $G' = (S, F)$, nous ajoutons tout sous-chemin reliant un noeud source ou un noeud ordianire à un noeud intermédiaire, et tout sous-chemin reliant un noeud intermédiaire à un noeud puits ou à un noeud ordinaire à condition de ne modifier ni la nature des noeuds de conflit ni l'ensemble $Col(\delta(i))$ pour chaque noeud intermédiaire $i \in S_2$. Ensuite nous ajoutons les épines aux noeuds intermédiaires. L'algorithme 3.5 décrit la procédure de liftage et l'algorithme 3.6 décrit l'ajout des épines. Notons que nous utilisons les mêmes procédures pour lifter le sous-multigraphe $G' = (S, F)$ avec les arcs de flux nul, en remplaçant toutefois la condition $X_{ij}^k > 0$ par $X_{ij}^k = 0$.

Algorithme 3.5 LiftPosCoupeImpaire($H = (V', A'), G' = (S, F)$)

Soit $H = (V', A')$,
Soit $G' = (S, F)$, où $S = S_1 \cup S_2$,
pour chaque arc $(i, j, k) \in A' \backslash F$ **tel que** $X_{ij}^k > 0$ et $i \in S$ et $j \in S$ **faire**
  si $\delta_F^+(i) \neq \emptyset$ et $\delta_F^-(j) \neq \emptyset$ **alors**
    si $i \in S_2$ et $k \in Col(\delta_F(i))$ **alors**
      $F := F \cup \{(i, j, k)\}$
    finsi
    si $j \in S_2$ et $k \in Col(\delta_F(j))$ **alors**
      $F := F \cup \{(i, j, k)\}$
    finsi
    si $i \in S_1$ et $\delta_F^-(i) \neq \emptyset$ et $k \in Col(\delta_F^+(i)) \cup (K \setminus Col(\delta_F(i)))$ **alors**
      $F := F \cup \{(i, j, k)\}$
    finsi
    si $j \in S_1$ et $\delta_F^+(j) \neq \emptyset$ et $k \in Col(\delta_F^-(j)) \cup (K \setminus Col(\delta_F(j)))$ **alors**
      $F := F \cup \{(i, j, k)\}$,
    finsi
  finsi
finpour

Proposition 3.5.6 *Soit $G' = (S, F)$ un sous-multigraphe partiel de $G = (V, A)$, où l'ensemble des sommets S est partitionné en S_1, S_2 et S_3 et l'ensemble des arcs F est partitionné en F_1 et F_2. G' est un sous-multigraphe conflictuel épineux.*

Démonstration.

$G' = (S, F)$ est un sous-multigraphe conflictuel épineux s'il vérifie les conditions de la définition 3.2.2. Par la proposition 3.5.4, nous avons démontré que le sous-multigraphe $G' = (S_1 \cup S_2, F_1)$ est un sous-multigraphe conflictuel non épineux, donc les quatres premières conditions de la définition 3.2.2 sont vérifiées (même après lifting). L'algorithme 3.6 permet

Algorithme 3.6 EpinePosCoupeImpaire($G = (V, A), G' = (S, F)$)

Soit $G = (V, A)$,
Soit $G' = (S, F)$, où $S = S_1 \cup S_2$,
$F_1 := F$
$F_2 := \emptyset$, $S_3 := \emptyset$,
pour chaque sommet $i \in S_2$ **faire**
  pour chaque arc $(j, i, k) \in \delta_{A\setminus F}(i)$ **tel que** $X^k_{ji} > 0$ **faire**
    si $k \notin Col(\delta^+_F(i))$ **alors**
      $F := F \cup \{(j, i, k)\}$, $F_2 := F_2 \cup \{(j, i, k)\}$, $S_3 := S_3 \cup \{j\}$,
      $Col(\delta^-_F(i)) := Col(\delta^-_F(i)) \cup \{k\}$
    finsi
  finpour
  pour chaque arc $(i, j, k) \in \delta_{A\setminus F}(i)$ **tel que** $X^k_{ij} > 0$ **faire**
    si $k \notin Col(\delta^-_F(i))$ **alors**
      $F := F \cup \{(i, j, k)\}$, $F_2 := F_2 \cup \{(i, j, k)\}$, $S_3 := S_3 \cup \{j\}$,
      $Col(\delta^+_F(i)) := Col(\delta^+_F(i)) \cup \{k\}$
    finsi
  finpour
finpour
$F := F_1 \cup F_2$, $S := S_1 \cup S_2 \cup S_3$.

l'ajout des épines au niveau de chaque noeud intermédiaire, donc la dernière condition de la définition 3.2.2 est aussi vérifiée. Nous pouvons conclure que $G' = (S, F)$ est un sous-multigraphe conflictuel épineux. □

3.5.5 Algorithme de séparation des inégalités de coupe impaire

À l'aide de la solution fractionnaire du MDVSP et du sous-multigraphe partiel orienté, $H = (V', A')$, nous construisons le sous-multigraphe partiel des chemins réguliers $G^* = (V', \mathcal{P}'_2)$. Ensuite, nous formulons et résolvons le problème ($CoupeImp$) à l'aide de la méthode de séparation et évaluation progressive. Lors de la résolution, nous récupérons toute solution réalisable de valeur strictement inférieure à 1. Pour chaque solution, nous construisons $G' = (S, F)$ un sous-multigraphe partiel. Nous enrichissons chaque sous-multigraphe partiel par des arcs de flux non nul, nous ajoutons les épines pour chaque sommet intermédiaire ; ensuite, nous enrichissons par des arcs de flux nul. Nous écrivons l'inégalité valide relative à G', si la violation est strictement positive, alors nous l'ajoutons au problème à résoudre. La procédure de séparation des inégalités de coupe impaire est décrite par l'algorithme 3.7.

Par la proposition 3.5.7, nous présentons la relation entre la capacité d'une coupe impaire séparée par le problème auxiliaire et la violation de l'inégalité valide correspondante. Cette relation justifie pourquoi nous retenons seulement les solutions réalisables de valeur plus

petite que 1.

Proposition 3.5.7 *Toute coupe impaire de capacité* $W^* = \sum_{p \in \mathcal{P}_1} w_p$ *inférieure à* 1 *correspond à une inégalité valide pour le MDVSP de violation inférieure ou égale à* $\frac{1 - W^*}{2}$.

Démonstration.

Selon la proposition 3.2.5, l'inégalité $\sum_{(i,j,k) \in F} X_{ij}^k \leq \left\lfloor \frac{|S_1|}{2} \right\rfloor + |S_2|$ est valide pour $(MDSVP)$. Sa violation est $\sum_{(i,j,k) \in F} X_{ij}^k - \left\lfloor \frac{|S_1|}{2} \right\rfloor - |S_2| = \sum_{(i,j,k) \in F} X_{ij}^k - \frac{|S_1|}{2} - |S_2| + \frac{1}{2}$ et la capacité d'une coupe impaire est égale à $W^* = \sum_{p \in \mathcal{P}_1} w_p = \sum_{s \in S_1} \sum_{(i,j,k) \in \delta_{A \setminus F}(s)} X_{ij}^k$.

Au niveau de chaque noeud $s \in S_1$ la relation suivante est vérifiée :

$$\sum_{(i,j,k) \in \delta_F(s)} X_{ij}^k + \sum_{(i,j,k) \in \delta_{A \setminus F}(s)} X_{ij}^k = 1 \tag{3.32}$$

En sommant sur tous les sommets appartenant à S_1, nous obtenons :

$$\sum_{s \in S_1} \sum_{(i,j,k) \in \delta_F(s)} X_{ij}^k + \sum_{s \in S_1} \sum_{(i,j,k) \in \delta_{A \setminus F}(s)} X_{ij}^k = |S_1| \tag{3.33}$$

Au niveau de chaque noeud $i \in S_2$, la relation suivante est aussi vérifiée (voir la démonstration de la proposition 3.2.5)

$$\sum_{(i,j,k) \in \delta_{F_1}(s)} X_{ij}^k + \sum_{(i,j,k) \in \delta_{F_2}(s)} 2X_{ij}^k \leq 2 \tag{3.34}$$

En sommant sur tous les sommets appartenant à S_2, nous obtenons :

$$\sum_{s \in S_2} \sum_{(i,j,k) \in \delta_{F_1}(s)} X_{ij}^k + \sum_{s \in S_2} \sum_{(i,j,k) \in \delta_{F_2}(s)} 2X_{ij}^k \leq 2|S_2| \tag{3.35}$$

Des relations (3.33) et (3.35), nous obtenons l'inégalité suivante :

$$\sum_{(i,j,k)\in F} X^k_{ij} + \frac{1}{2}\sum_{s\in S_1}\sum_{(i,j,k)\in\delta_{A\setminus F}(s)} X^k_{ij} \leq \frac{|S_1|}{2} + |S_2| \quad (3.36)$$

$$\Rightarrow \sum_{(i,j,k)\in F} X^k_{ij} + \frac{1}{2}W^* \leq \frac{|S_1|}{2} + |S_2| \quad (3.37)$$

$$\Rightarrow \sum_{(i,j,k)\in F} X^k_{ij} - \frac{|S_1|}{2} - |S_2| \leq -\frac{1}{2}W^* \quad (3.38)$$

$$\Rightarrow violation - \frac{1}{2} \leq -\frac{1}{2}W^* \quad (3.39)$$

$$\Rightarrow violation \leq \frac{1-W^*}{2} \quad (3.40)$$

□

Algorithme 3.7 Séparation d'inégalités de Coupe Impaire($Solution, ListInégValid$)

1: extraire la solution de la relaxation linéaire et construire un sous-multigraphe partiel orienté, $H = (V', A')$
2: Soit $\mathcal{P}$ la famille de tous les chemins réguliers $P_{i,j}$ reliant le noeud i au noeud j dans H.
3: résoudre le problème ($CoupeImp$) à l'aide de Cplex.
4: **pour** toute solution réalisable de valeur < 1 **faire**
5: extraire S_1 l'ensemble des sommets $i \in V$ tel que $x_i = 1$
6: extraire $\mathcal{P}_2$ l'ensemble des chemins $p \in \mathcal{P}$ **tels que** l'intersection de p avec S_1 est de cardinalité 2.
7: extraire $\mathcal{P}'_2$ l'ensemble des sous-chemins $p' = \{i_k, i_{k+1}, i_{k+2}, ..., i_l\}$ de $p \in \mathcal{P}_2$ **tels que** $x_{i_k} = x_{i_l} = 1$ et $x_{i_{k+1}} = x_{i_{k+2}} = ... = x_{i_{l-1}} = 0$
8: extraire $F'_{p'}$ l'ensemble des arcs du sous-chemin p'
9: extraire $I_{p'} = \{i_{k+1}, i_{k+2}, ..., i_{l-1}\}$ l'ensemble des sommets intermédiaires du sous-chemin p'.
10: Soit $S_2 = \bigcup_{p'\in\mathcal{P}'_2} I_{p'}$
11: Soit $S = S_1 \cup S_2$
12: Soit $F = \bigcup_{p'\in\mathcal{P}'_2} F'_{p'}$
13: construire $G' = (S, F)$ un sous-multigraphe partiel de G^*,
14: LiftPosCoupeImpaire($A, G' = (S, F)$)
15: EpineCoupeImpaire($A, G' = (S, F)$)
16: LiftNulCoupeImpaire($A, G' = (S, F)$)
17: EpineNulCoupeImpaire($A, G' = (S, F)$)
18: ajouter l'inégalité valide relative à G' à $ListInégValid$
19: **finpour**

3.6 Méthodes d'élimination de variables

Dans cette section, nous présentons les principes d'élimination de variables et les outils nécessaires pour leur application au MDVSP. Notons que nous appliquons deux méthodes d'élimination de variables sans génération de colonnes, soit une méthode directe (section 3.6.1) et une méthode bidirectionnelle (section 3.6.2).

3.6.1 Méthode directe

Afin d'éliminer des variables (c'est-à-dire fixer leur valeur à 0) du $(MDVSP)$, nous avons appliqué le critère de la proposition 2.4.1 en utilisant la solution de la relaxation linéaire du $(MDVSP)$ et une solution réalisable obtenue à l'aide de l'heuristique décrite par l'algorithme 3.8. Cette dernière consiste à transformer le problème du MDVSP en un problème d'horaires de véhicules avec un seul dépôt SDVSP. Nous remplaçons l'ensemble des dépôts par un seul dépôt, qui est représenté par le noeud $n+1$. Le dépôt est relié à chaque noeud i, pour tout $i = 1, 2, \ldots, n$, du réseau par un arc de coût $c_{n+1,i} = \min_{k \in K} c_{n+k,i}$. Chaque noeud i, pour tout $i = 1, 2, \ldots, n$, est relié au dépôt par un arc de coût $c_{i,n+1} = \min_{k \in K} c_{i,n+k}$. Après la résolution du $(SDVSP)$, nous choisissons p_1 variables possédant les plus petits coûts réduits. Nous créons le $(MDVSP)$ relatif au p_1 variables sélectionnées et nous résolvons sa relaxation linéaire. En utilisant le même critère, nous sélectionnons p_2 variables, nous créons le $(MDVSP)$ relatif et nous utilisons la méthode de séparation et évaluation progressive pour résoudre le problème.

Algorithme 3.8 Heuristique (d'A. Hadjar)

1: Construire le SDVSP
2: Résoudre le $(SDVSP)$
3: Trier les variables selon l'ordre croissant des coûts réduits
4: Sélectionner les p_1 premières variables, et créer le $(MDVSP)$ réduit, nommé $MDVSP\_R1$
5: Résoudre la relaxation linéaire du $MDVSP\_R1$
6: Trier les variables selon l'ordre croissant des coûts réduits
7: Sélectionner les p_2 premières variables, et créer le $(MDVSP)$ réduit, nommé $MDVSP\_R2$
8: Résoudre $MDVSP\_R2$
9: Nommer la solution trouvée S_{Heur} et la valeur de cette solution UB_{Heur}

Dans le but d'obtenir une solution heuristique en peu de temps, nous limitons les tailles des problèmes $MDVSP\_R1$ et $MDVSP\_R2$, c'est-à-dire les valeurs de p_1 et p_2 ne doivent pas être très grandes. Soient m le nombre total d'arcs et n le nombre total de noeuds. Soit $NbCotSP$ (respectivement $NbCotR1$) le nombre de variables du $SDVSP$ (respectivement

$MDVSP\_R1$) dont le coût réduit est plus petit que 0.1. Les valeurs de p_1 et p_2 sont définies comme suit :

$VarMax1 = \max\{10n+1, \lfloor \frac{m}{10} \rfloor + 1\}$, $p_1 = \max\{VarMax1, NbCotSP\}$

$VarMax2 = \max\{n+1, \lfloor \frac{m}{10} \rfloor + 1\}$, $p_1 = \max\{VarMax2, NbCotR1\}$

3.6.2 Méthode bidirectionnelle

Soit $(\pi, \rho^1, \rho^2, ..., \rho^{|K|})$ une solution réalisable duale du $(MDVSP)$, où $\pi = (\pi_1, ..., \pi_n, ..., \pi_{n+|K|})$ est le vecteur des variables duales associées aux contraintes (2.2) et (2.3) et $\rho^k = (\rho^k_1, \rho^k_2, ..., \rho^k_n)$ est le vecteur des variables duales associées aux contraintes (2.4). De plus, soit $r(\pi, \rho)^T = c^T - \pi A - \sum_{k \in K} \rho^k D^k$ le vecteur des coûts réduits, et UB une borne supérieure pour la valeur optimale du $(MDVSP)$.

Selon la proposition 2.4.1, si $r_{ij}(\pi, \rho) > UB - \sum_{i=1}^{i=n} \pi_i - \sum_{k \in K} v_k \pi_{n+k} - \sum_{k \in K} \rho^k d^k$, alors $X^k_{ij} = 0$ dans toute solution optimale du $(MDVSP)$. Ceci peut s'écrire : $lb_{ij}(\pi, \rho) = r_{ij}(\pi, \rho) + \sum_{i=1}^{i=n} \pi_i + \sum_{k \in K} v_k \pi_{n+k} + \sum_{k \in K} \rho^k d^k > UB$.

Vecteur des variables duales et calcul des coûts réduits

Soit $\tilde{c}_{ij}$ le coût associé à chaque arc $(i, j, k) \in A$, tel que

$$\tilde{c}_{ij} = \begin{cases} c_{ij} - \pi_i & \text{si } i = 1, 2, ..., n \\ c_{(k+n)j} - \pi_{k+n} & \text{si le noeud } i \text{ représente le dépôt } D_k \ \forall k \in K \end{cases}$$

Nous avons utilisé l'algorithme de recherche du plus court chemin pour les graphes acycliques pour calculer les longueurs des plus courts chemins suivantes :

- $\ell_t^{k\rightarrow}$ représente la longueur du plus court chemin du dépôt D_k au dépôt D_k, qui est égal à 0,
- $\ell_i^{k\rightarrow}$ représente la longueur du plus court chemin du dépôt D_k à i, et
- $\ell_j^{k\leftarrow}$ représente la longueur du plus court chemin du dépôt D_k à j en inversant le sens des arcs.

En utilisant l'équation (2.24), la plus grande valeur du coût réduit de la variable X^k_{ij} par rapport à π est égale à :

$$\bar{r}^k_{ij}(\pi) = \tilde{c}_{ij} + \ell_i^{k\rightarrow} + \ell_j^{k\leftarrow}. \tag{3.41}$$

Par conséquent, la proposition suivante nous permet d'éliminer des arcs (ou fixer des variables) du $(MDVSP)$.

Proposition 3.6.1 *Si* $\bar{r}_{ij}^k(\pi) > UB - \sum_{i=1}^{n} \pi_i - \sum_{k \in K} v_k \pi_{n+k}$, *alors* $X_{ij}^k = 0$ *dans toute solution optimale du* $(MDVSP)$, *où* UB *représente une borne supérieure sur la valeur optimale de la solution du* $(MDVSP)$.

Élimination d'arcs après l'ajout des coupes

Après l'ajout des inégalités valides, le côut réduit associé à chaque arc $(i, j, k) \in A$ devient :

$$\tilde{c}_{ij} = \begin{cases} c_{ij} - \pi_i - \sum_{t=1}^{p} a_{ijk}^t \pi_t & \text{si } i = 1, 2, ..., n \\ c_{n+kj} - \pi_{k+n} - \sum_{t=1}^{p} a_{n+kjk}^t \pi_t & \text{si le noeud } i \text{ représente le dépôt } D_k \end{cases}$$

où :

- π_t représente la variable duale associée à l'inégalité valide t, $\forall t = 1, ..., p$,
- a_{ij}^t représente le coefficient de la variable X_{ij}^k dans l'inégalité t, $\forall t = 1, ..., p$.

De ce qui précède, nous constatons que l'élimination de variables dépend principalement des valeurs des deux bornes inférieure et supérieure. Alors, nous avons appliqué une méthode d'élimination de variables en deux phases. Dans la première, nous avons appliqué les méthodes directe et bidirectionnelle en utilisant une borne supérieure fournie par l'heuristique (algorithme 3.8) et la valeur de la relaxation linéaire comme borne inférieure. Dans la deuxième phase, nous avons soumis à Cplex le programme linéaire en nombres entiers obtenu dans la première phase et la valeur de la solution heuristique comme solution initiale. Si au noeud racine, la borne supérieure est meilleure que celle de l'heuristique, alors nous appliquons la méthode bidirectionnelle encore une fois. La procédure d'élimination de variables est décrite par l'algorithme 3.9.

3.7 Algorithme de résolution du MDVSP

Au noeud racine de l'arbre de branchement, l'algorithme débute par la résolution de la relaxation linéaire du $(MDVSP)$. Si le problème relaxé possède une solution optimale et si elle n'est pas entière, alors l'algorithme applique la méthode d'élimination de variables, présentée dans la section 3.6. Ensuite, l'algorithme utilise les méthodes de séparation des inégalités valides décrites par les algorithmes 3.3, 3.4 et 3.7. S'il existe des plans coupants, alors l'algorithme ajoute les inégalités valides au programme linéaire et le résout de nouveau

Algorithme 3.9 Méthode d'élimination de variables

1: Déterminer une solution heuristique et une borne supérieure UB_{Heur}
2: Résoudre la relaxation linéaire du $(MDVSP)$ et récupérer la solution duale
3: Éliminer des variables en appliquant la méthode directe d'élimination de variables
4: Éliminer des variables en appliquant la méthode bidirectionnelle
5: Résoudre le programme linéaire résultant en nombres entiers, en limitant le nombre de noeud à 1 et récupérer la borne supérieure donnée par l'heuristique de CPLEX
6: **si** Au noeud racine, CPLEX fournit une borne supérieure meilleure que UB_{Heur} **alors**
7: Éliminer des variables en appliquant la méthode bidirectionnelle
8: **finsi**

(au niveau du même noeud). La même opération est répétée jusqu'à qu'il n'existe plus de plans coupants ; dans ce cas nous appliquons une autre fois la méthode d'élimination de variables. Par la suite, nous continuons la résolution du programme linéaire en nombres entiers, où deux noeuds fils sont créés. Dans l'un, une variable X_{ij}^k est fixée à 0, et dans l'autre, la même variable est fixée à 1. Le choix de la variable de branchement est fait par le logiciel utilisé pour la résolution du problème. Au niveau de chaque noeud de l'arbre de branchement, l'algorithme procède de la même façon. Si la solution optimale existe et n'est pas entière, alors la méthode de séparation est appliquée jusqu'à ce qu'il n'existe plus de plans coupants ; dans ce cas, deux noeuds fils sont créés et ainsi de suite jusqu'à ce qu'il n'existe plus de noeuds actifs dans l'arbre de branchement. La méthode de résolution du MDVSP est décrite par l'algorithme 3.10.

Algorithme 3.10 Résolution du MDVSP

1: Au noeud racine, appliquer la méthode d'élimination de variables en utilisant l'algorithme 3.9
2: Au noeud racine, appliquer la méthode de séparation des inégalités valides
3: **tant que** Il existe des inégalités violées au noeud racine **faire**
4: Ajouter les inégalités violées au problème
5: Résoudre la relaxation linéaire du problème
6: Appliquer la méthode de séparation des inégalités
7: **fintant que**
8: Appliquer la méthode d'élimination de variables en utilisant l'algorithme 3.9
9: Continuer la résolution du programme linéaire en nombres entiers à l'aide de Cplex.

En considérant les différentes combinaisons possibles de nos procédures, nous avons testé plusieurs stratégies. Nous présentons dans le tableau 3.1 les caractéristiques des stratégies choisies. Les colonnes du tableau indiquent dans l'ordre de gauche à droite : le numéro de la stratégie, si le pré-traitement de Cplex est autorisé ou non, si les coupes de Cplex sont ajoutées ou non, si la procédure d'élimination de variables est appliquée ou non, si au noeud racine les inégalités de cycle impair, les inégalités de coupe impaire ou les inégalités de clique

de cardinalité 3 sont ajoutées ou non, ensuite si elles sont ajoutées aux autres noeuds de l'arbre de branchement.

Tableau 3.1 Caractéristiques des stratégies

N° Stratégie	Pré-traitement	Coupes Cplex	VF	Au noeud racine			Aux noeuds fils		
				Cy.Imp	Cp.Imp	K 3	Cy.Imp	Cp.Imp	K 3
0	✓	✓							
1			✓						
2			✓	✓					
3			✓		✓				
4			✓			✓			
5			✓	✓	✓	✓			
6			✓	✓	✓	✓	✓		
7			✓		✓	✓	✓		
8				✓					
9				✓	✓	✓			

3.8 Application et résultats

Pour tester notre algorithme, nous avons utilisé un générateur aléatoire des instances du MDVSP qui a été créé par Carpaneto et *al.* (1989). Ce générateur a été utilisé par plusieurs chercheurs (Ribeiro et Soumis (1994), Fischetti et *al.*(2001), Hadjar et *al.* (2006) ainsi que d'autres) pour tester leur algorithme. Les résultats présentés dans nos tableaux sont des moyennes sur 5 instances différentes générées de type A ou B, pour un nombre donné de tâches et de dépôts. Par exemple, pour un problème de 400 tâches avec 4 dépôts, le nombre de noeuds est égal à la moyenne des nombres de noeuds de 5 instances différentes de 400 tâches avec 4 dépôts. Nous avons fixé une limite de temps de résolution à 48 heures. Nous avons utilisé CPLEX 12.4.0.0 pour résoudre nos problèmes sur la machine Briarée du Réseau Calcul Québec de Haute Performance (RQCHP). Briarée possède 630 noeuds, chacun contient 2 processeurs Intel (Intel Xeon X5650 2.66GHz Westmere-EP). Notons que nous n'avons pas utilisé la version parallèle de CPLEX.

3.8.1 Comparaison de stratégies

Afin de déterminer la meilleure stratégie, nous comparons les 10 stratégies présentées dans le tableau 3.1, en utilisant 5 instances aléatoires de type A avec 600 tâches et 5 dépôts et 5 instances de type A avec 800 tâches et 5 dépôts. Les résultats de cette comparaison sont

présentés dans le tableau 3.2. Les colonnes du tableau indiquent dans l'ordre de gauche à droite : le numéro de la stratégie, pour les instances 600A5 (resp. 800A5) : le pourcentage des variables fixées à la racine, le nombre total d'inégalités valides ajoutées et le temps CPU total de résolution. Nous constatons que le pourcentage moyen des variables fixées est de 93% pour les instances de 600 tâches et de 78% pour les instances de 800 tâches. Nous remarquons aussi que l'ajout des inégalités valides permet de supprimer plus de variables, ce qui implique que les plans coupants améliorent la valeur de la borne inférieure. L'importance de l'élimination de variables se concrétise en observant la différence entre les temps CPU enregistrés par les stratégies 2 et 5 par rapport aux temps CPU enregistrés par les stratégies 8 et 9. Les meilleurs temps CPU sont enregistrés par les stratégies 2 et 5. Nous retenons les stratégies 0, 2 et 5 pour faire une analyse des profils de performance.

Tableau 3.2 Comparaison des stratégies

	600A5			800A5		
Stratégie	VF %	Nb.Inégalités valides	Temps CPU (s)	VF %	Nb.Inégalités valides	Temps CPU (s)
0	0.00	0.0	1470.92	0.00	0.0	6041.70
1	92.44	0.0	630.84	78.23	0.0	6615.60
2	93.70	28.0	480.14	78.79	30.8	3358.3
3	93.74	6.2	620.76	78.33	1.8	6307.9
4	92.96	0.4	625.52	78.23	0.2	9428.7
5	93.02	33.8	511.60	78.22	41.1	2985.5
6	94.38	94.8	682.42	78.22	131.0	5238.9
7	93.14	70.8	621.10	78.33	90.0	6146.5
8	0.00	27.2	1716.44	0.00	29.2	9714.2
9	0.00	28.4	1673.22	0.00	37.4	11428.9

3.8.2 Analyse des profils de performance

Afin de comparer les trois stratégies sélectionnées suite à une première comparaison, nous utilisons l'analyse des profils de performance introduite par Dolan et Moré (2002). Nous présentons en premier le principe de cette technique.

Soit S l'ensemble des stratégies tel que $|S| = n_s$ représente le nombre de stratégies et P l'ensemble des problèmes à tester tel que $|P| = n_p$ représente le nombre de problèmes. Nous désirons utiliser le temps CPU de résolution comme une mesure de performance.

Soit t_{ps} le temps CPU total requis pour la résolution du problème p en utilisant la stratégie s, pour tout problème p et stratégie s. Soit r_{ps} le ratio de performance tel que

$$r_{ps} = \frac{t_{ps}}{\min\limits_{s \in S}\{t_{ps}\}}. \tag{3.42}$$

Soit $P_\tau^s = \{p \in P | r_{ps} \leq \tau\}$, pour tout $s \in S$, pour tout scalaire $\tau \in \mathbb{R}$, $\tau \geq 1$. La performance de la stratégie $s \in S$ est donnée par la fonction suivante :

$$\rho_s(\tau) = \frac{|P_\tau^s|}{n_p}, \quad \forall \tau \in \mathbb{R}. \tag{3.43}$$

Par exemple, $\rho_s(1.5) = 0.8$ indique que 80% des instances peuvent être résolues par la stratégie s en au plus 150% du meilleur temps enregistré par toutes les stratégies.

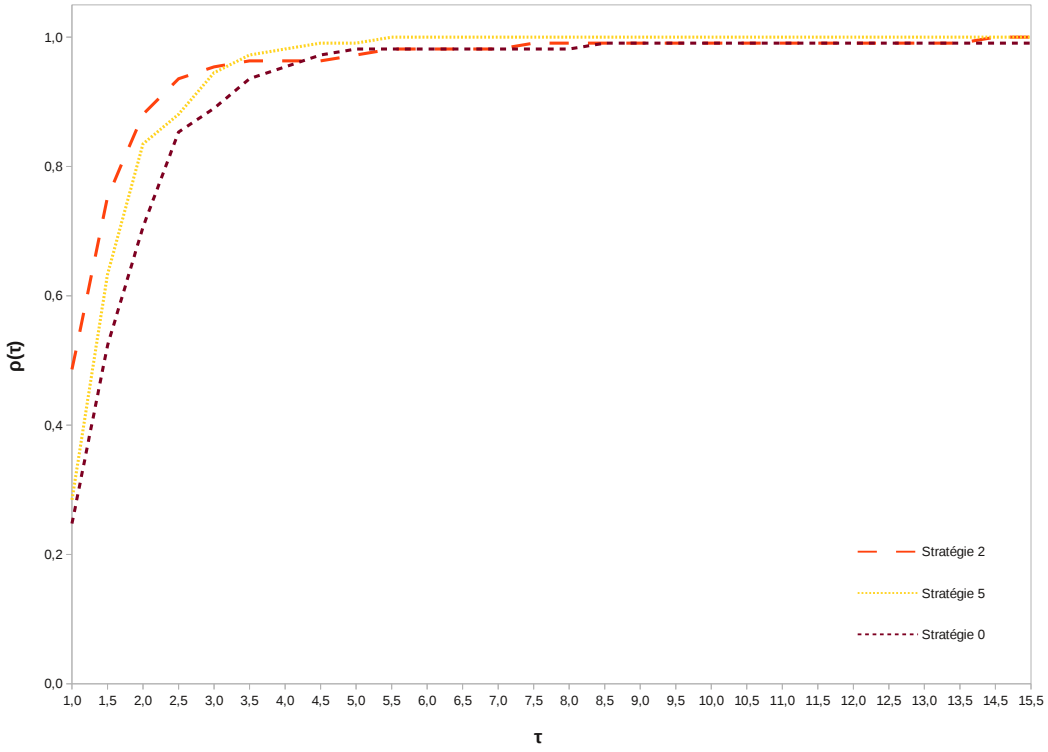

Figure 3.12 Courbes des profils de performance.

Nous utilisons 130 instances générées aléatoirement, en considérant des instances de type A et B, de 4 et 5 dépôts et le nombre des tâches varie entre 400 et 1000. Les courbes des profils de performance des stratégies 0, 2 et 5 sont illustrées à la figure 3.12. La comparaison des courbes des profils de performance montre que la stratégie 2 est plus performante pour

les petites valeurs de τ. Par contre c'est la stratégie 5 qui converge le plus rapidement.

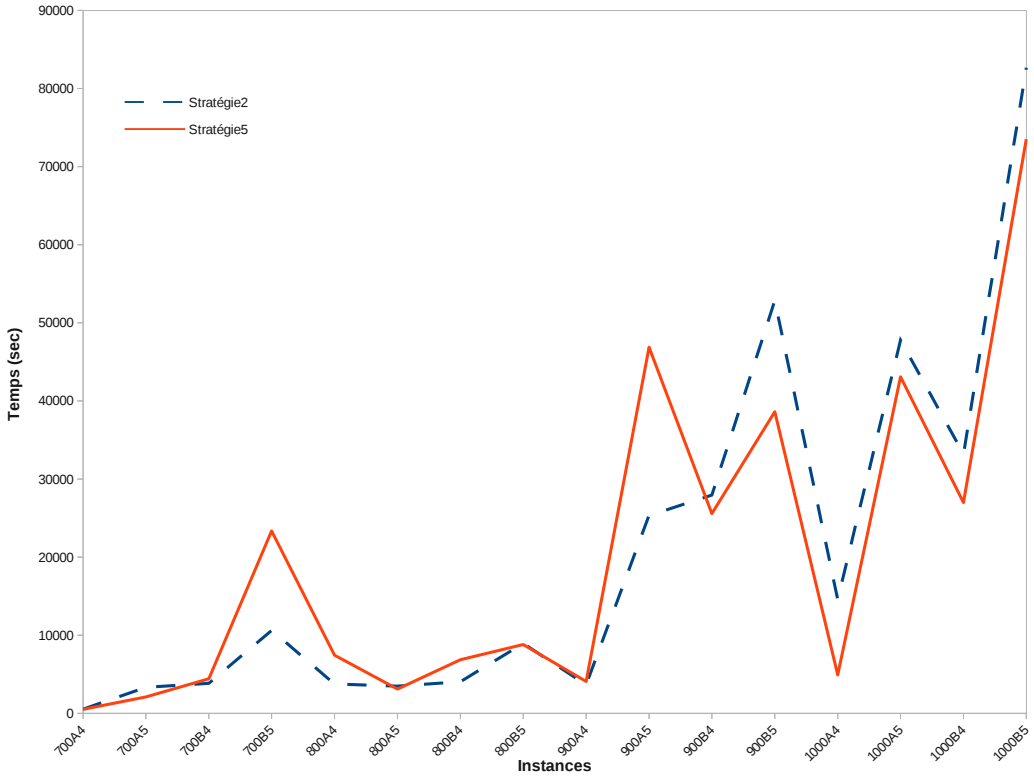

Figure 3.13 Comparaison des temps de résolution des stratégies 2 et 5.

De la comparaison des courbes des profils de performance, il semble qu'il n'existe pas un grand écart entre les stratégies 2 et 5. Afin de vérifier si l'ajout des inégalités de coupe impaire est utile ou non, nous comparons les temps de résolution des stratégies 2 et 5. La comparaison est représentée par la figure 3.13. Nous remarquons que pour les instances de taille 800 et moins, la stratégie 2 est plus rapide que la stratégie 5. Par contre pour les instances de taille 900 et 1000 la stratégie 5 est meilleure. Nous concluons que, pour les grandes instances, l'ajout des inégalités de coupe impaire aide à réduire les temps de résolution.

3.8.3 Quelques statistiques

Dans le tableau 3.3, nous présentons quelques statistiques. Les colonnes du tableau indiquent de gauche à droite : le nom de l'instance, le nombre total de variables avant élimination, le nombre de variables fixées en pourcentage en appliquant la stratégie 2 et le nombre de variables fixées en pourcentage en appliquant la stratégie 5. Notons que nous présentons des instances de type A seulement. Pour les instances de type B, les grandeurs sont différentes

mais les conclusions sont les mêmes. Nous n'observons pas de grandes différences entre les deux stratégies. Nous constatons qu'en moyenne, nous éliminons entre 62.6% et 98.5% des variables au noeud racine. Les pourcentages sont plus élevés pour les instances de 600 tâches et moins, ce qui est peut-être dû à la qualité de la borne supérieure.

Les moyennes des nombres d'inégalités valides ajoutées par chaque stratégie ainsi que le nombre de noeuds de l'arbre de branchement sont présentées dans le tableau 3.4. Les colonnes du tableau indiquent dans l'ordre de gauche à droite : le nom de l'instance, pour la stratégie 2 : le nombre d'inégalités de cycle impair ajoutées, le nombre de noeuds de l'arbre de branchement, pour la stratégie 5 : le nombre d'inégalités de cycle impair ajoutées, le nombre d'inégalités de clique de cardinalité 3 ajoutées, le nombre d'inégalités de coupe impaire ajoutées, le nombre total d'inégalités valides ajoutées et le nombre de noeuds de l'arbre de branchement. Nous constatons que nous avons identifié plus d'inégalités valides avec la stratégie 5 qu'avec la stratégie 2. Nous remarquons qu'en appliquant la stratégie 5 nous avons identifié plus d'inégalités de cycle impair que celles de coupe impaire à la racine (il faut noter que nous ne considérons pas les structures identiques), ce qui n'est pas surprenant, vu que les structures des coupes impaires sont plus grandes que les structures de cycles impairs. De plus il est possible que le problème auxiliaire arête la résolution avant de trouver beaucoup d'inégalités de coupe impaire, puisque nous avons limité le nombre de noeuds de son arbre de branchement. Nous remarquons aussi, qu'en moyenne le nombre d'inégalités de clique de cardinalité 3 identifiées (qui sont des cycles impairs de longueur 3) est inférieur à 1. Ce n'est pas surprenant étant donné que les structures de cycles impairs conflictuels ordinaires (sans épine) sont rares dans le multigraphe du MDVSP. Généralement, les arbres de branchement obtenus par l'application de la stratégie 5 contiennent moins de noeuds.

Nous avons mentionné dans la section 3.5 qu'une solution entière du problème auxiliaire peut fournir une coupe impaire qui correpond à une inégalité violée, une coupe paire ou une coupe impaire qui correpond à une inégalité valide non-violée. Le tableau 3.5 contient les statistiques des solutions du problème auxiliaire en nombres entiers utilisé pour la séparation des inégalités de coupe impaire dans la stratégie 5. Nous constatons que la proportion d'inégalités de coupe impaire ajoutées varie entre 31.9% et 48.5%. Par contre, la proportion de coupes paires est inférieure à 10.0%, sauf pour les instances 400A5 et 500A4 dont les proportions sont 21.9% et 11.0%, respectivement. Ce résultat est assez bon surtout que nous résolvons un seul problème auxiliaire, à chaque itération, pour séparer toutes les inégalités de coupe impaire.

Tableau 3.3 Nombre de variables fixées dans les stratégies 2 et 5.

Instance	NbVar	NbVarFix (%) Stratégie 2	NbVarFix (%) Stratégie 5
400A4	194452.0	98.5	98.4
400A5	245253.0	86.1	87.6
500A4	303904.0	96.5	95.0
500A5	379906.0	86.6	92.4
600A4	433015.2	96.2	94.2
600A5	548417.0	93.7	94.4
700A4	595441.6	90.7	90.7
700A5	747081.0	62.7	62.6
800A4	781039.2	76.9	76.9
800A5	969901.0	78.8	78.2
900A4	969084.8	91.2	91.2
900A5	1231693.0	85.6	85.7
1000A4	1213471.6	77.8	77.6
1000A5	1518470.0	92.9	92.9

Tableau 3.4 Nombre de coupes ajoutées dans les stratégies 2 et 5.

	Stratégie 2		Stratégie 5				
Instance	NbTotal Cy.Imp	Nb.Noeuds branch	NbTotal Cy.Imp	NbTotal K 3	NbTotal Cp.Imp	NbTotal Ing.Valid	Nb.Noeuds branch
400A4	14.8	74.4	16.0	0.0	6.6	22.6	88.6
400A5	14.2	729.4	13.8	0.0	5.0	18.8	612.0
500A4	30.0	592.2	32.2	0.2	6.6	39.0	937.6
500A5	16.4	744.6	22.8	0.6	5.4	28.8	380.0
600A4	34.4	562.6	31.8	0.2	3.6	35.6	425.0
600A5	28.0	6133.4	26.6	0.6	4.6	31.8	7546.6
700A4	31.8	3637.6	32.2	0.0	4.4	36.6	1356.4
700A5	32.6	34686.8	28.6	0.0	4.4	33.0	13453.2
800A4	39.2	30543.0	39.6	0.0	5.2	44.8	59090.6
800A5	30.8	26246.4	35.4	0.2	5.8	41.4	18167.8
900A4	24.8	32643.2	24.8	0.4	2.8	28.0	37977.4
900A5	31.0	165496.8	36.6	0.2	6.4	43.2	300869.0
1000A4	31.75	71439.3	31.0	0.3	4.5	35.75	34021.0
1000A5	37.0	487212.75	33.75	0.3	6.75	40.75	129831.0

Tableau 3.5 Les solutions du problème auxiliaire en nombres entiers dans la stratégie 5.

Instance	Nb Cp.Imp	% Cp.Imp	Nb Cp.Pair	% Cp.Paire	NbTotal Cp.Trouvée
400A4	6.6	48.5	0.6	4.4	13.6
400A5	5.0	39.1	2.8	21.9	12.8
500A4	6.6	36.3	2.0	11.0	18.2
500A5	5.4	37.5	1.4	9.7	14.4
600A4	3.6	32.1	0.8	7.1	11.2
600A5	4.6	37.7	1.2	9.8	12.2
700A4	4.4	31.9	1.2	8.7	13.8
700A5	4.4	37.3	0.4	3.4	11.8
800A4	5.2	36.1	0.6	4.2	14.4
800A5	5.8	37.7	0.6	3.9	15.4
900A4	2.8	26.4	0.4	3.8	10.6
900A5	6.4	33.7	0.4	2.1	19
1000A4	4.5	39.1	0.25	2.2	11.5
1000A5	6.75	40.3	1.25	7.5	16.75

3.9 Conclusion

Dans le présent chapitre, nous avons présenté un algorithme combinant les méthodes d'élinimation de variables, de séparation et d'évaluation progressive et de génération de plans coupants pour la résolution du MDVSP, en utilisant la formulation mathématique de multi-flot dans les réseaux proposée par Ribeiro et Soumis (1994).

En premier, nous avons généralisé la notion de sous-multigraphe épineux proposé par Hadjar *et al.* (2006). Ensuite, nous avons exploité les points communs entre le MDVSP et les problèmes de couplage et de stable pour établir des relations entre les inégalités d'Edmonds (1965) pour le problème de couplage, les inégalités de trous impairs pour le problème de stable et les inégalités valides pour le MDVSP. En utilisant ces relations, nous avons réussi à identifier deux structures particulières, que nous avons nommées les cycles impairs conflictuels et les coupes impaires. Nous avons proposé un algorithme polynomial pour la séparation de la première et nous avons formulé le problème de séparation des inégalités de coupe impaire comme un programme linéaire en nombres entiers dont la complexité n'est pas encore connue. Notons que jusqu'à maintenant nous n'avons pas la preuve que le problème de séparation est NP-complet ou non, malgré que nous croyons qu'il l'est.

Nous avons montré que l'ajout des inégalités valides basées sur les cycles impairs conflictuels et les coupes impaires permet d'améliorer les temps de résolution des instances du

MDVSP. Des résultats obtenues, nous avons remarqué qu'en moyenne le temps de séparation des inégalités valides représente 30% du temps total de résolution du problème. De plus, le temps de séparation des inégalités de cycle impair représente 7.3% du temps total de séparation. Par contre le temps de séparation des inégalités de coupe impaire représente 92.6% du temps total de séparation et la séparation des inégalités de clique de cardinalité 3 ne représente que 0.1% du temps total de séparation. Ces constatations permettent de justifier le fait qu'il n'existe pas une grande différence entre les stratégies 2 et 5 pour les instances de moins de 800 tâches. C'est pour cette raison, nous pensons qu'il est préférable de proposer une méthode heuristique pour la séparation de cette catégorie d'inégalités valides. Nous avons élaboré une recherche tabou qui n'a permis d'identifier que des structures de petite taille et donc très peu d'inégalités violées.

CHAPITRE 4

INÉGALITÉS VALIDES ET ALGORITHMES DE SÉPARATION POUR LE SPP

4.1 Introduction

Dans le présent chapitre, nous étudions le SPP pur. Nous considérons une matrice $A = (a_{ij})$ dont les coefficients sont égaux à 0 ou 1. Soit $I = \{1, \ldots, m\}$ l'ensemble des lignes et $J = \{1, \ldots, n\}$ l'ensemble des colonnes de la matrice A. Soit $c_j, j \in J$, le coût de la colonne $j \in J$. L'élément a_{ij} est égal à 1 si la colonne $j \in J$ couvre la ligne $i \in I$ et 0 sinon. Le SPP consiste à déterminer un sous-ensemble de colonnes $N \subseteq J$, de coût minimal, tel que chaque ligne $i \in I$ est couverte exactement par une colonne $j \in N$. Notons que nous utilisons la formulation mathématique du problème de partitionnement présentée dans la section 2.2

Notre objectif est de résoudre le SPP en appliquant un algorithme utilisant la méthode de séparation et évaluation progressive combinée aux méthodes d'élimination de variables et de génération de plans coupants. Nous retrouvons dans la littérature les inégalités de cycle impair et les inégalités de clique qui ont été proposées pour le SPP, en considérant un graphe simple, où l'ensemble des sommets représente les colonnes du problème et il existe une arête reliant deux sommets du graphe si les deux colonnes associées couvrent une même ligne. Sans considérer un tel graphe, nous prouvons qu'il existe deux nouvelles classes d'inégalités valides propres au SPP et nous présentons la relation entre les deux classes. Notons aussi que nous pouvons associer un multigraphe au SPP, où les sommets représentent les lignes, et il existe une arête de couleur j reliant deux sommets si et seulement si la colonne j contribue aux lignes représentées par les deux sommets.

Dans la section 4.2, nous décrivons les deux nouvelles classes d'inégalités valides et nous prouvons qu'il existe une relation entre les deux. Dans la section 4.3, nous proposons une méthode de séparation de chaque famille d'inégalités valides proposées. Les problèmes de séparation sont formulés comme des programmes linéaires en nombres entiers. Dans la section 4.4, nous proposons de résoudre un autre problème auxilaire pour séparer les inégalités de clique. Dans la section 4.5, nous présentons les détails de la méthode de résolution du problème de partitionnement. Dans la section 4.6, nous utilisons des instances de Hoffman et Padberg (1993) et d'autres instances aléatoires générées à l'aide du générateur proposé par Lewis *et al.* (2008) pour tester la performance de notre algorithme comparativement à un logiciel commercial. Nous terminons par une conclusion dans la section 4.7.

4.2 Nouvelles inégalités valides pour le SPP

Dans cette section, nous introduisons deux nouvelles classes d'inégalités valides pour le SPP. Avant d'énoncer les propositions et le théorème démontrant la relation entre les deux classes d'inégalités, nous présentons un exemple afin d'éclaircir le principe.

Exemple 4.2.1 *Soit $A = (a_{ij})_{i\in I, j\in J}$ une matrice dont les coefficents sont égaux à 0 ou 1. Soit I l'ensemble des lignes de A et J l'ensemble des colonnes de A.*

Soit $A^* = (a^*_{ij})_{i\in S, j\in J}$ une sous-matrice de A, où $S \subseteq I$ est un sous-ensemble de lignes de cardinalité impaire. Dans cet exemple, nous choisissons $|S| = 5$. L'ensemble des colonnes J est partitionné en trois sous-ensembles J_0, J_1 et J_2 tels que :

- *l'intersection de chaque colonne $j \in J_2$ avec les lignes de S contient un nombre pair d'éléments non nuls, c'est-à-dire, $\sum_{i\in S} a_{ij} = 2l_j$ où $l_j \in \mathbb{N}^*$,*
- *l'intersection de chaque colonne $j' \in J_1$ avec les lignes de S contient un nombre impair d'éléments non nuls, c'est-à-dire, $\sum_{i\in S} a_{ij'} = 2l_{j'} + 1$ où $l_{j'} \in \mathbb{N}$,*
- *l'intersection de chaque colonne $j'' \in J_0$ avec les lignes de S ne contient aucun élément non nul, c'est-à-dire, $\sum_{i\in S} a_{ij''} = 0$.*

La sous-matrice A^* ressemble alors à

$$
\begin{array}{c}
 \\
1 \\
2 \\
3 \\
4 \\
5
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccc}
j_1 & j_2 & \cdots & j_{|J_2|} & j'_1 & j'_2 & \cdots & j'_{|J_1|} & j''_1 & j''_2 & \cdots & j''_{|J_0|}
\end{array} \\
\left(\begin{array}{cccccccccccc}
1 & 0 & \ldots & 1 & 0 & 1 & \ldots & 0 & 0 & 0 & \ldots & 0 \\
1 & 0 & \ldots & 1 & 0 & 1 & \ldots & 1 & 0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & 1 & 1 & 1 & \ldots & 1 & 0 & 0 & \ldots & 0 \\
0 & 1 & \ldots & 1 & 0 & 1 & \ldots & 0 & 0 & 0 & \ldots & 0 \\
0 & 1 & \ldots & 0 & 0 & 1 & \ldots & 1 & 0 & 0 & \ldots & 0
\end{array}\right)
\end{array}
$$

Pour couvrir les lignes de S, nous pouvons choisir parmi les colonnes de J_2, soit une colonne telle que $\sum_{i\in S} a_{ij} = 4$ ou $\sum_{i\in S} a_{ij} = 2$, soit deux colonnes telles que $\sum_{i\in S} a_{ij} = 2$ pour chaque colonne, ou bien aucune colonne, ce qui implique que $\frac{1}{2}\sum_{j\in J_2}\sum_{i\in S} a_{ij}x_j \leq 2$ est valide pour le SPP. Les colonnes de J_2 permettent donc de couvrir au plus quatre lignes de S. Afin de couvrir la (ou les) ligne(s) restante(s), nous devons choisir au moins une colonne appartenant à J_1, c'est-à-dire, $\sum_{j'\in J_1} x_{j'} \geq 1$ est aussi valide pour le SPP.

$$
\begin{array}{c}
\\
1\\
2\\
3\\
4\\
5
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccc}
j_1 & j_2 & \dots & j_{|J_2|} & j'_1 & j'_2 & \dots & j'_{|J_1|} & j''_1 & j''_2 & \dots & j''_{|J_0|}
\end{array}\\
\left(\begin{array}{cccccccccccc}
1 & 0 & \dots & 1 & 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0\\
1 & 0 & \dots & 1 & 0 & 1 & \dots & 1 & 0 & 0 & \dots & 0\\
0 & 0 & \dots & 1 & 1 & 1 & \dots & 1 & 0 & 0 & \dots & 0\\
0 & 1 & \dots & 1 & 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0\\
0 & 1 & \dots & 0 & 0 & 1 & \dots & 1 & 0 & 0 & \dots & 0
\end{array}\right)
\end{array}
$$

$$
\frac{1}{2}\sum_{j\in J_2}\sum_{i\in S} a_{ij}x_j \leq 2 \qquad \sum_{j'\in J_1} x_{j'} \geq 1
$$

Nous pouvons généraliser le principe présenté dans l'exemple 4.2.1 afin d'introduire deux nouvelles classes d'inégalités valides propres au SPP. Les propositions 4.2.2 et 4.2.3 les définissent et nous démontrons la relation entre celles-ci dans le théorème 4.2.4.

Considérons une sous-matrice $A^* = (a^*_{ij})_{i\in S, j\in J}$ de la matrice $A = (a_{ij})_{i\in I, j\in J}$, où $S \subseteq I$ représente l'ensemble des lignes de A^*. L'ensemble des colonnes J est partitionné en J_0, J_1 et J_2. La sous-matrice A^* est dite impaire si les conditions suivantes sont vérifiées

- A^* contient un nombre impair de lignes, c'est-à-dire, $|S| = 2k+1$, $k \in \mathbb{N}^*$,
- chaque colonne $j \in J_2$ contient un nombre pair d'éléments non nuls, c'est-à-dire, $\forall j \in J_2, \sum_{i\in S} a_{ij} = 2l_j$ où $l_j \in \mathbb{N}^*$,
- chaque colonne $j \in J_1$ contient un nombre impair d'éléments non nuls, c'est-à-dire, $\forall j \in J_1, \sum_{i\in S} a_{ij} = 2l_j + 1$ où $l_j \in \mathbb{N}$,
- chaque colonne $j \in J_0$ ne contient aucun élément non nul, c'est-à-dire, $\forall j \in J_0$, $\sum_{i\in S} a_{ij} = 0$.

Proposition 4.2.2 *Soit A^* une sous-matrice impaire. L'inégalité*

$$
\sum_{j\in J_2} l_j x_j \leq \left\lfloor \frac{|S|}{2} \right\rfloor \tag{4.1}
$$

est valide pour le SPP.

Démonstration. Des contraintes (2.9), on déduit

$$\sum_{j\in J} a_{ij}x_j = 1 \qquad \forall i \in S \tag{4.2}$$

$$\Rightarrow \sum_{i\in S}\sum_{j\in J} a_{ij}x_j = |S| \tag{4.3}$$

$$\Rightarrow \sum_{i\in S}\sum_{j\in J_2} a_{ij}x_j + \sum_{i\in S}\sum_{j\in J\setminus J_2} a_{ij}x_j = |S| \tag{4.4}$$

$$\Rightarrow \sum_{i\in S}\sum_{j\in J_2} a_{ij}x_j \leq |S| \tag{4.5}$$

$$\Rightarrow \sum_{j\in J_2} 2l_j x_j \leq |S| \tag{4.6}$$

$$\Rightarrow \sum_{j\in J_2} l_j x_j \leq \frac{|S|}{2} \tag{4.7}$$

Par conséquent, puisque $l_j \in \mathbb{N}^*$ et $x_j \in \mathbb{N}$, $\forall j \in J_2$, l'inégalité $\sum_{j\in J_2} l_j x_j \leq \left\lfloor \frac{|S|}{2} \right\rfloor$ est valide pour le SPP. □

Proposition 4.2.3 *Soit A^* une sous-matrice impaire et $x = (x_j)_{j\in J}$ une solution entière du SPP. L'inégalité $\sum_{j\in J_1} x_j \geq 1$ est valide pour le SPP si et seulement si l'inégalité $\sum_{j\in J_2} l_j x_j \leq \left\lfloor \frac{|S|}{2} \right\rfloor$ est valide pour le SPP.*

Démonstration. Supposons que $\sum_{j\in J_2} l_j x_j \leq \left\lfloor \frac{|S|}{2} \right\rfloor$.

$$\Longleftrightarrow \sum_{j\in J_2} l_j x_j \leq \frac{|S|-1}{2} \tag{4.8}$$

$$\Longleftrightarrow \sum_{j\in J_2} 2l_j x_j \leq |S|-1 \tag{4.9}$$

$$\Longleftrightarrow \sum_{j\in J_2} 2l_j x_j \leq \sum_{j\in J_2} 2l_j x_j + \sum_{j\in J_1}(2l_j+1)x_j + \sum_{j\in J_0}(0)x_j - 1 \tag{4.10}$$

$$\Longleftrightarrow \quad 1 \leq \sum_{j\in J_1}(2l_j+1)x_j \tag{4.11}$$

De l'inégalité (4.11), nous pouvons conclure qu'il existe au moins une colonne $j \in J_1$ telle que $x_j \neq 0$. Par conséquent, l'inégalité $\sum_{j\in J_1} x_j \geq 1$ est valide pour le SPP.

Supposons maintenant que $\sum_{j\in J_1} x_j \geq 1$, ce qui implique que $\sum_{j\in J_1}(2l_j+1)x_j \geq 1$. Selon les inégalités (4.8) à (4.11), c'est équivalent à $\sum_{j\in J_2} l_j x_j \leq \left\lfloor \frac{|S|}{2} \right\rfloor$. □

Par les propositions 4.2.2 et 4.2.3, nous avons prouvé que les deux familles d'inégalités sont valides pour toute solution entière du SPP. Mais la question qui s'impose est la suivante : lorsqu'une inégalité d'une des deux familles est violée par une solution fractionnaire, est-ce que l'inégalité de l'autre famille est aussi violée ? La réponse à cette question est donnée par le théorème 4.2.4.

Théorème 4.2.4 *Soit A^* une sous-matrice impaire et $\bar{x} = (\bar{x}_j)_{j\in J}$ une solution fractionnaire du SPP. Si $\sum_{j\in J_2} l_j \bar{x}_j > \left\lfloor \frac{|S|}{2} \right\rfloor$, alors $\sum_{j\in J_1} \bar{x}_j < 1$.*

Démonstration. Supposons que $\sum_{j\in J_2} l_j \bar{x}_j > \left\lfloor \frac{|S|}{2} \right\rfloor$, alors

$$\sum_{j\in J_2} l_j \bar{x}_j > \frac{|S|-1}{2} \tag{4.12}$$

$$\Rightarrow \sum_{j\in J_2} 2l_j \bar{x}_j > |S|-1 \tag{4.13}$$

On sait que

$$\sum_{j\in J_2} 2l_j \bar{x}_j + \sum_{j\in J_1}(2l_j+1)\bar{x}_j + \sum_{j\in J_0}(0)\bar{x}_j = |S| \tag{4.14}$$

De l'inégalité (4.13) et l'équation (4.14), on déduit

$$\sum_{j\in J_2} 2l_j \bar{x}_j > \sum_{j\in J_2} 2l_j \bar{x}_j + \sum_{j\in J_1}(2l_j+1)\bar{x}_j - 1 \tag{4.15}$$

$$\text{d'où} \qquad 1 > \sum_{j\in J_1}(2l_j+1)\bar{x}_j \tag{4.16}$$

Puisque $\sum_{j\in J_1}(2l_j+1)\bar{x}_j \geq \sum_{j\in J_1} \bar{x}_j$, alors on peut déduire que

$$1 > \sum_{j\in J_1} \bar{x}_j \tag{4.17}$$

□

Nous avons donc démontré que si une solution fractionnaire viole l'inégalité définie par les colonnes de J_2, alors l'inégalité définie par les colonnes de J_1 est aussi violée. Mais l'inverse n'est pas forcément vrai comme le prouve l'exemple 4.2.5.

Exemple 4.2.5 *Soit la sous-matrice A^* formée de 5 lignes et soit $\bar{x} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, 0, \frac{1}{4}, \frac{1}{4})$ une solution fractionnaire du problème de partitionnement.*

$$
\begin{array}{c}
 \\
1 \\
2 \\
3 \\
4 \\
5
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc}
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4}
\end{array} \\
\left(\begin{array}{cccccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}\right)
\end{array}
$$

Nous constatons que $\sum_{j \in J_2} \sum_{i \in S} a_{ij} x_j = 1 < \frac{5-1}{2}$. Donc l'inégalité introduite par la proposition 4.2.2 n'est pas violée par la solution fractionnaire $\bar{x}$. Par contre, l'inégalité introduite par la proposition 4.2.3 est violée, puisque $\sum_{j' \in J_1} x_{j'} = \frac{1}{2} < 1$.

4.3 Séparation des nouvelles inégalités valides

Pour la séparation des inégalités valides définies par les propositions 4.2.2 et 4.2.3, nous proposons d'utiliser la solution fractionnaire courante du SPP pour construire un problème auxiliaire afin de séparer chaque classe d'inégalités valides proposées. La résolution de ce problème auxiliaire, permet d'identifier une sous-matrice impaire, qui vérifie les conditions des deux propositions. Dans cette section, nous présentons les détails de chaque méthode. Nous nommons les inégalités définies par la proposition 4.2.2 « type I » et celles définies par la proposition 4.2.3 « type II ».

Soit $\bar{x} = (\bar{x}_j)_{j \in J}$ une solution fractionnaire du SPP. Dans ce qui suit, nous affectons à chaque colonne $j \in J$ le poids $\bar{x}_j$. Notons que nous considérons seulement l'ensemble des colonnes dont le poids est fractionnaire noté $\bar{J} = \{j \in J | 0 < \bar{x}_j < 1\}$ afin d'identifier les inégalités valides. Nous les enrichissons, ensuite, en ajoutant les variables de valeur nulle. Par conséquent, nous retenons seulement les lignes couvertes par les colonnes de $\bar{J}$.

4.3.1 Méthode de séparation des inégalités de type I

Notre objectif est de déterminer une sous-matrice impaire $A^* = (a^*_{ij})_{i\in S, j\in J_0\cup J_1\cup J_2}$, telle que le poids total des colonnes appartenant à J_2, soit maximal.

Nous associons à chaque ligne $i \in I$ une variable binaire w_i, qui prend la valeur 1 si la ligne i appartient à l'ensemble S, et 0 sinon. À chaque colonne $j \in \bar{J}$, nous associons une variable binaire z_j et une variable entière l_j. La variable z_j prend la valeur 1 si l'intersection de la colonne j avec l'ensemble S est de cardinalité impaire, et 0 sinon. Quant à la variable l_j, elle est égale à $\left\lfloor \frac{1}{2}\sum_{i\in S} a_{ij} \right\rfloor$. À chaque élément de la matrice A, nous associons une variable binaire m_{ij}. Cette dernière est égale à a_{ij} si $j \in J_2$ et $i \in S$, et 0 sinon. Le problème auxiliaire de séparation des inégalités de type I se formule comme un programme linéaire en nombres entiers.

$$
\text{(PbAuxiI)} \begin{cases}
\max \quad \displaystyle\sum_{j\in\bar{J}}\sum_{i\in I} \frac{1}{2}\bar{x}_j m_{ij} - k & & (4.18)\\
\displaystyle\sum_{i\in I} w_i = 2k+1 & & (4.19)\\
\displaystyle\sum_{i\in I} a_{ij}w_i - z_j - 2l_j = 0 & \forall j \in \bar{J} & (4.20)\\
m_{ij} \le a_{ij} w_i & \forall i \in I,\ \forall j \in \bar{J} & (4.21)\\
m_{ij} \le a_{ij}(1 - z_j) & \forall i \in I,\ \forall j \in \bar{J} & (4.22)\\
m_{ij} \in \{0,1\} & \forall i \in I,\ \forall j \in \bar{J} & (4.23)\\
w_i \in \{0,1\} & \forall i \in I & (4.24)\\
z_j \in \{0,1\} & \forall j \in \bar{J} & (4.25)\\
l_j \in \mathbb{Z}, l_j \geqslant 0 & \forall j \in \bar{J} & (4.26)\\
1 \le k \le k_{max}, k \in \mathbb{Z} & & (4.27)
\end{cases}
$$

L'objectif (4.18) maximise la violation de l'inégalité de type I. La contrainte (4.19) assure que la cardinalité de S soit impaire. Les contraintes (4.20) permettent de calculer les valeurs de l_j et de z_j. Les contraintes (4.21) et (4.22) assurent que $m_{ij} = a_{ij}$, si $i \in S$ et $j \in J_2$ et $m_{ij} = 0$ sinon. Les contraintes (4.23), (4.24) et (4.25) indiquent que les variables m_{ij}, w_j et z_j sont binaires. Finalement les contraintes (4.26) et (4.27) définissent le domaine des variables l_j et k. Précisons que le paramètre k_{max} permet de limiter la taille de l'ensemble S, dans le but d'accélérer la résolution du problème. Nous avons choisi $k_{max} = \left\lfloor \frac{|I|}{2} \right\rfloor - 1$.

Proposition 4.3.1 *Toute solution réalisable du problème (PbAuxiI) de valeur strictement supérieure à* 0 *induit une inégalité de type I violée par la solution fractionnaire du SPP.*

Démonstration.

Soit une solution réalisable (m, w, l, z) du problème (PbAuxiI) qui définit les ensembles suivants

- $S = \{i \in I \mid \ w_i = 1\}$,
- $J_2 = \{j \in \bar{J} \mid l_j > 0 \ \text{ et } z_j = 0\}$,
- $J_1 = \{j \in \bar{J} \mid l_j \geq 0 \text{ et } z_j = 1\}$,
- $J_0 = \{j \in \bar{J} \mid l_j = 0 \text{ et } z_j = 0\}$.

La valeur de cette solution est $\sum_{j \in \bar{J}} \sum_{i \in I} \frac{1}{2} \bar{x}_j m_{ij} - k$.

Il s'ensuit que :

- $|S| = \sum_{i \in S} w_i = 2k + 1$, ceci est assuré par la contrainte (4.19).
- $\forall j \in J_2$ et $\forall i \in S$, $m_{ij} = a_{ij}$, ceci est assuré par les contraintes (4.21) et (4.22). Par conséquent, $\sum_{i \in S} m_{ij} = \sum_{i \in S} a_{ij} = 2l_j$, $\forall j \in J_2$,
- $\forall j \in J_1$ et $\forall i \in S$, $m_{ij} = 0$, ceci est assuré par la contrainte (4.22). Et par la contrainte (4.20) nous avons $\sum_{i \in S} a_{ij} = 2l_j + 1$, $\forall j \in J_1$,
- $\forall j \in J_0$ et , $\forall i \in S$, $m_{ij} = a_{ij} = 0$, ceci est assuré par les contraintes (4.20) et (4.22). Et par conséquent, $\sum_{i \in S} a_{ij} = 0 \ \forall j \in J_0$,
- Nous déduisons donc que la valeur de la solution est $\sum_{j \in J_2} l_j \bar{x}_j - \frac{|S| - 1}{2}$.

Par conséquent la sous-matrice $A^* = (a_{ij})_{i \in S, j \in J_0 \cup J_1 \cup J_2}$ est une sous-matrice impaire. Si de plus, $\sum_{j \in J_2} \bar{x}_j l_j - \frac{|S| - 1}{2} > 0$, alors l'inégalité de type I est violée par la solution fractionnaire du SPP. □

Notons que nous n'avons pas besoin de la solution optimale du problème auxiliaire, mais seulement d'une solution réalisable de valeur strictement supérieure à 0. Afin d'identifier le plus grand nombre d'inégalités de type I, nous récupérons toute solution réalisable trouvée lors de la résolution du problème auxiliaire de séparation (PbAuxiI). Pour chaque solution de valeur strictement supérieure à 0, nous extrayons les ensembles S et J_2, ainsi que les valeurs des variables l_j et nous écrivons l'inégalité valide définie par la proposition 4.2.2. Étant donné que nous avons considéré seulement les variables non entières, c'est-à-dire, telles que $0 < x_j < 1$, nous enrichissons l'inégalité valide par les variables nulles. Ensuite, nous ajoutons l'inégalité valide au problème à résoudre. La procédure de séparation est décrite par l'algorithme 4.1.

Algorithme 4.1 Séparation d'inégalités valides de Type I($Solution, ListInégValid$)

extraire la solution fractionnaire du SPP
résoudre le problème (PbAuxiI)
pour toute solution réalisable de valeur > 0 **faire**
  extraire la valeur de k
  extraire S l'ensemble des lignes $i \in I$ telles que $w_i = 1$
  extraire J_2 l'ensemble des colonnes $j \in \bar{J}$ telles que $l_j > 0$ et $z_j = 0$
  pour chaque colonne $j \in J \backslash \bar{J}$ telle que $\bar{x}_j = 0$ **faire**
    si $\sum_{i \in S} a_{ij}$ est pair **alors**
      $J_2 := J_2 \cup \{j\}$
      $l_j := \frac{1}{2} \sum_{i \in S} a_{ij}$.
    finsi
  finpour
  ajouter l'inégalité $\sum_{j \in J_2} l_j x_j \leq \left\lfloor \frac{|S|}{2} \right\rfloor$ à $ListInégValid$
finpour

4.3.2 Méthode de séparation des inégalités de type II

Notre but est d'identifier une sous-matrice impaire $A^* = (a^*_{ij})_{i \in S, j \in J_0 \cup J_1 \cup J_2}$, telle que le poids total des colonnes appartenant à J_1 soit minimal.

Nous associons à chaque ligne $i \in I$ une variable binaire w_i, qui prend la valeur 1 si la ligne i appartient à l'ensemble S, et 0 sinon. À chaque colonne $j \in \bar{J}$, nous associons une variable binaire z_j et une variable entière l_j. La variable z_j prend la valeur 1 si l'intersection de la colonne j avec l'ensemble S est de cardinalité impaire, et 0 sinon. La variable l_j est égale à la valeur $\left\lfloor \frac{1}{2} \sum_{i \in S} a_{ij} \right\rfloor$.

Le problème auxiliaire de séparation des inégalités de type II se formule comme un programme linéaire en nombres entiers.

$$
\text{(PbAuxiII)} \begin{cases}
\min \quad \sum_{j \in \bar{J}} \bar{x}_j z_j & & (4.28) \\
\sum_{i \in I} w_i = 2k+1 & & (4.29) \\
\sum_{i \in I} a_{ij} w_i - z_j - 2l_j = 0 & \forall j \in \bar{J} & (4.30) \\
w_i \in \{0,1\} & \forall i \in I & (4.31) \\
z_j \in \{0,1\} & \forall j \in \bar{J} & (4.32) \\
l_j \in \mathbb{Z}, l_j \geqslant 0 & \forall j \in \bar{J} & (4.33) \\
1 \leq k \leq k_{max}, k \in \mathbb{Z} & & (4.34)
\end{cases}
$$

L'objectif (4.28) minimise le poids total des colonnes ayant un nombre impair de coefficients non nuls dans la sous-matrice impaire A^*. La contrainte (4.29) assure que la cardinalité de S soit impaire. Les contraintes (4.30) permettent de calculer les valeurs de l_j et z_j. Les contraintes (4.31) et (4.32) indiquent que les variables w_j et z_j sont binaires. Finalement, les contraintes (4.33) et (4.34) indiquent que les variables l_j et k sont entières. Notons que k_{max} est un paramètre fixé qui permet de réduire la taille de l'ensemble S ; nous l'avons fixé à $\left\lfloor \frac{|I|}{2} \right\rfloor - 1$.

Proposition 4.3.2 *Toute solution réalisable du problème (PbAuxiII) de valeur strictement inférieure à* 1 *induit une inégalité de type II violée par la solution fractionnaire du SPP.*

Démonstration.

Soit une solution réalisable (w, l, z) du problème (PbAuxiII) qui définit les ensembles suivants

- $S = \{i \in I \mid \ w_i = 1\}$,
- $J_2 = \{j \in \bar{J} \mid l_j > 0 \ \text{ et } z_j = 0\}$,
- $J_1 = \{j \in \bar{J} \mid l_j \geq 0 \text{ et } z_j = 1\}$,
- $J_0 = \{j \in \bar{J} \mid l_j = 0 \text{ et } z_j = 0\}$.

La valeur de cette solution est $\sum_{j \in \bar{J}} \bar{x}_j z_j$.

En considérant les contraintes de (4.29) à (4.30), nous pouvons déduire que :

- $|S| = \sum_{i \in S} w_i = 2k+1$,
- $\forall j \in J_2, \sum_{i \in S} a_{ij} = 2l_j$,
- $\forall j \in J_1, \sum_{i \in S} a_{ij} = 2l_j + 1$,

- $\forall j \in J_0, \sum_{i \in S} a_{ij} = 0$,
- Par conséquent, la valeur de la solution est $\sum_{j \in J_1} \bar{x}_j$.

Nous constatons que la sous-matrice $A^* = (a^*_{ij})_{i \in S, j \in J_0 \cup J_1 \cup J_2}$ est une sous-matrice impaire. De plus, si $\sum_{j \in J_1} \bar{x}_j < 1$, alors nous avons identifié une inégalité de type II violée par la solution fractionnaire du SPP. □

Lors de la résolution du problème (PbAuxiII), nous récupérons toute solution réalisable de valeur strictement inférieure à 1. Pour chaque solution, nous extrayons les ensembles S et J_1 et nous écrivons l'inégalité valide définie par la proposition 4.2.3. Ensuite, nous enrichissons l'inégalité valide trouvée par les variables nulles. Finalement, nous ajoutons l'inégalité valide au problème à résoudre. La procédure de séparation est décrite par l'algorithme 4.2.

Algorithme 4.2 Séparation d'inégalités valides de Type II($Solution, ListInégValid$)

1: extraire la solution fractionnaire du SPP
2: résoudre le problème (PbAuxiII)
3: **pour** toute solution réalisable de valeur < 1 **faire**
4: extraire S l'ensemble des lignes $i \in I$ telles que $w_i = 1$
5: extraire J_2 l'ensemble des colonnes $j \in J$ telles que $z_j = 1$
6: **pour** chaque colonne $j \in J$ telle que $x_j = 0$ **faire**
7: **si** $\sum_{i \in S} a_{ij} = 2l_j + 1$ **alors**
8: $J_2 := J_2 \cup \{j\}$,
9: **finsi**
10: **finpour**
11: ajouter l'inégalité $\sum_{j \in J_1} x_j \geq 1$ à $ListInégValid$
12: **finpour**

4.4 Séparation des inégalités de clique

Dans cette section, nous proposons une méthode de séparation des inégalités de clique introduites par Padberg (1973) (voir le théorème 2.2.2). Notre méthode consiste à utiliser la solution fractionnaire pour identifier un sous-graphe complet, à l'aide de la résolution d'un programme auxiliaire en nombres entiers.

Soit $G = (V, E)$ un graphe simple non orienté, où chaque sommet $j \in V$ représente la colonne $j \in J$, et il existe une arête $(j, j') \in E$ si et seulement si les colonnes j et j' ont au moins une ligne en commun. De plus, le poids de chaque sommet $j \in V$ est la valeur de la variable $\bar{x}_j$. Soit la matrice $B = (b_{ej})_{e \in E, j \in V}$, où $b_{ej} = 1$ si le sommet j est l'une des extrémités de l'arête e et 0 sinon.

Notre objectif est d'identifier un sous-graphe complet K du graphe G, tel que :

- le nombre de sommets dans K est égal à k, où k est une constante positive prédéfinie,
- le nombre d'arêtes est égal à $\dfrac{k(k-1)}{2}$,
- le degré de chaque sommet est $(k-1)$,
- le poids total des sommets, est maximal.

Nous associons à chaque sommet $j \in V$ une variable binaire y_j qui prend la valeur 1 si le sommet j appartient au sous-graphe K, et 0, sinon. Nous associons à chaque arête $e \in E$ une variable binaire w_e qui prend la valeur 1 si les deux extrémités de l'arête e appartiennent à K, et 0 sinon. Le problème de séparation d'une clique de cardinalité k se formule comme un programme linéaire en nombres entiers.

$$
(P_k)\begin{cases}
\max \quad \displaystyle\sum_{j\in V} \bar{x}_j y_j & & (4.35)\\
\displaystyle\sum_{j\in V} y_j = k & & (4.36)\\
\displaystyle\sum_{e\in E} w_e = \frac{k(k-1)}{2} & & (4.37)\\
\displaystyle\sum_{e\in E} b_{ej} w_e - (k-1) y_j = 0 & \forall j \in V & (4.38)\\
w_e \in \{0,1\} & \forall e \in E & (4.39)\\
y_j \in \{0,1\} & \forall j \in V & (4.40)
\end{cases}
$$

L'objectif (4.35) maximise le poids total des sommets du sous-graphe K. La contrainte (4.36) garantit que le nombre de sommets du sous-graphe K est égal à k. La contrainte (4.37) assure que le sous-graphe K est complet, c'est-à-dire, que le nombre d'arêtes est égal à $\dfrac{k(k-1)}{2}$. Les contraintes (4.38) précisent que chaque sommet appartenant à K est adjacent à tous les autres sommets de K. Les contraintes (4.39) et (4.40) indiquent que les variables w_e et y_j sont binaires. Notons que, pour chaque valeur fixée de k, nous résolvons le programme (P_k).

Proposition 4.4.1 *Toute solution réalisable du problème (P_k) correspond à une clique de cardinalité k. De plus, si la valeur de la solution est strictement supérieure à 1, l'inégalité de clique (2.11) est violée.*

Démonstration. Soit une solution réalisable (y, w) du problème (P_k), qui définit le sous-graphe $K = (S, E(S))$, où

- $S = \{j \in V \mid y_j = 1\}$,

- $E(S) = \{e \in E \mid w_e = 1\}$,
- la valeur de la solution est $\sum_{j \in V} \bar{x}_j y_j$.

Il s'ensuit que,

- $|S| = \sum_{j \in S} y_j = k$, ceci est assuré par la contrainte (4.36),
- $|E(S)| = \sum_{e \in E(S)} w_e = \frac{k(k-1)}{2}$, ceci est assuré par la contrainte (4.37),
- $\forall j \in S$, $\sum_{e \in E(S)} b_{ej} = k - 1$, ceci est assuré par la contrainte (4.38).

Nous avons donc séparé un sous-graphe $K = (S, E(S))$ tel que :

- le nombre de sommets est $|S| = k$,
- le nombre d'arêtes est $|E(S)| = \frac{k(k-1)}{2}$, et
- le degré de chaque sommet de S est $(k-1)$.

Par définition le sous-graphe $K = (S, E(S))$ est une clique de cardinalité k. De plus, si $\sum_{j \in S} \bar{x}_j > 1$, alors nous avons détecté une inégalité de clique qui est violée par la solution fractionnaire $\bar{x}$. □

Nous résolvons plusieurs problèmes (P_k) à l'aide de la méthode de séparation et évaluation progressive. Les valeurs de k varient entre 3 et k_{max}, qui a été fixé à 10 pour nos tests. De plus, si pour $k < k_{max}$ le programme (P_k) n'identifie pas une inégalité valide, alors nous arrêtons la recherche de clique de cardinalité supérieure.

Pour chaque programme linéaire en nombres entiers (P_k), nous récupérons toute solution réalisable de valeur strictement supérieure à 1. Pour chaque solution, nous séparons l'ensemble des sommets S, nous écrivons l'inégalité de clique et nous l'ajoutons à la liste des inégalités valides. Notons que nous considérons seulement les variables non nulles. L'algorithme 4.3 décrit la procédure de séparation.

4.5 Algorithme de résolution du SPP

Nous proposons de résoudre le problème de partitionnement d'ensemble en combinant les méthodes d'élimination de variables, de séparation et évaluation progressive et de génération de plans coupants.

Afin de réduire la taille du problème, nous appliquons la méthode directe d'élimination de variables (voir 3.6.1), après la résolution de la relaxation linéaire du SPP et la récupération de sa valeur, ainsi que la valeur du coût réduit de chaque variable. Nous récupérons la valeur de la première solution réalisable fournie par Cplex. Si la valeur du coût réduit d'une variable x_j est inférieure à la différence entre la valeur de la solution réalisable et la valeur de la

Algorithme 4.3 Séparation Inégalités Clique($Solution, ListInégValid$)

1: extraire la solution fractionnaire du SPP
2: $k := 1$
3: **tant que** $k \leq k_{max}$ **faire**
4: résoudre le problème auxiliaire (P_k)
5: **si** il n'existe pas de solution réalisable de valeur > 1 **alors**
6: $k := k_{max} + 1$
7: **sinon**
8: **pour** toute solution réalisable de valeur > 1 **faire**
9: extraire S l'ensemble des sommets $j \in V$ tels que $y_j = 1$
10: ajouter l'inégalité $\sum_{j \in S} x_j \leq 1$ à $ListInégValid$
11: **finpour**
12: **finsi**
13: **fintant que**

relaxation linéaire, alors cette variable est éliminée du problème.

Ensuite, nous débutons la résolution du problème en nombres entiers obtenu après l'élimination de variables. Dans le but d'améliorer la valeur de la borne inférieure au noeud racine, nous ajoutons des plans coupants au noeud racine en appliquant les méthodes de séparation des inégalités valides décrites par les algorithmes 4.1, 4.2 et 4.3. S'il existe des inégalités violées par la solution fractionnaire, nous les ajoutons à l'ensemble des contraintes et nous résolvons le problème obtenu. La même opération est répétée jusqu'à qu'il n'existe plus d'inégalités valides. Dans ce cas, nous continuons la résolution du dernier programme linéaire en nombres entiers. L'algorithme 4.4 résume la méthode de résolution du SPP que nous proposons.

Malheureusement, nous avons constaté que l'algorithme 4.1 consomme beaucoup de temps pour détecter des inégalités violées. Nous avons donc décidé d'utiliser l'algorithme 4.2 pour séparer les inégalités valides de types II et l'algorithme 4.3 pour séparer les inégalités de clique. Notons aussi que nous cherchons toute clique de cardinalité variant entre 3 et 10, et s'il n'existe pas de clique de cardinalité k, alors nous arrêtons la recherche de clique de cardinalité supérieure.

4.6 Application et résultats

Pour tester la performamnce de notre algorithme, nous avons retenu 3 des 55 instances (sppaa01, appnw04 et appus01) testées par Hoffman et Padberg (1993). Nous avons supprimé les 52 autres instances, parce que Cplex version 12.4.0.0 a réussi à trouver la solution optimale en moins de 20 sec. Nous avons généré 55 instances aléatoires à l'aide du générateur aléatoire

Algorithme 4.4 Résolution du SPP

```
1: Résoudre la relaxation linéaire du SPP
2: Débuter la résolution du programme linéaire en nombres entiers du SPP
3: si le noeud courant = noeud racine alors
4:    appliquer la méthode directe d'élimination de variables
5:    appliquer la méthode de séparation des inégalités valides
6:    tant que Il existe des inégalités violées au noeud racine faire
7:       Ajouter les inégalités violées au problème
8:       Résoudre la relaxation linéaire du problème
9:       Appliquer les procédure de séparation des inégalités valides
10:   fintant que
11: finsi
12: Continuer la résolution du programme linéaire en nombres entiers
```

proposé par Lewis et *al.* (2008), où pour une taille donnée et une densité donnée, nous générons 5 instances aléatoires. Nous avons effectué nos tests sur des machines du GERAD (Groupe d'études et de recherche en analyse des décisions) ayant deux processeurs Opteron 250 à 2.4GHZ et 16G de mémoire. Notons que nous n'avons pas utilisé la version parallèle de CPLEX.

Nous présentons dans le tableau 4.1 les caractéristiques des instances utilisées. Nous indiquons dans l'ordre de gauche à droite : le nom de l'instance, le nombre de contraintes, le nombre de variables, la densité moyenne des colonnes (en %), la valeur de la relaxation linéaire et la valeur de la solution optimale. Notons que les résultats présentés sont les moyennes de 5 instances avec les mêmes caractéristiques à savoir le nombre de variables, le nombre de contraintes et la même densité.

4.6.1 Analyse des profils de performance

Nous avons effectué une analyse des profils de performance introduite par Dolan et Moré (2002) (les détails de cette analyse sont présentés dans le chapitre précédent) en utilisant les 55 instances aléatoires générées à l'aide du générateur de Lewis *et al.* (2008). Par cette analyse, nous avons comparé les temps CPU obtenus en résolvant les problèmes avec Cplex par défaut et avec deux stratégies en utilisant la méthode décrite par l'algorithme 4.4. La première stratégie consiste à ajouter les inégalités valides de type II seulement. Par contre, la deuxième stratégie consiste à ajouter les inégalités valides de type II ainsi que les inégalités de clique.

La figure 4.1 présente les courbes de profils de performance pour les instances aléatoires. L'analyse montre qu'il n'existe pas une grande différence entre nos deux stratégies, ce qui est normal, vu que nous n'avons pas identifié un grand nombre d'inégalités de clique. La compa-

Tableau 4.1 Les caractéristiques des instances

Instance	Nb Contraintes	Nb Variables	Densité %	Valeur. relax.lin	Valeur. Optimale
sppaa01	823	8904	1.0	55535.44	56137.00
sppnw04	36	87482	20.2	16310.67	16862.00
sppus01	145	1053137	9.1	9963.07	10036.00
Pb0	300	1300	3.0	285.34	409.20
Pb1	250	650	4.0	1548.04	2485.60
Pb2	200	500	6.0	1146.14	2510.20
Pb3	1500	1650	12.0	43751.01	131688.00
Pb4	300	2300	16.0	17.35	3526.80
Pb5	1500	2000	23.0	13285.78	116002.00
Pb6	2300	300	34.0	7.05	4009.80
Pb7	400	1400	58.0	7.34	3335.60
Pb8	300	1300	59.0	5.76	2434.00
Pb9	200	1400	62.0	3.67	1509.40
Pb10	1300	300	82.0	3.84	1061.20

raison des courbes de profils montre aussi que dans certains cas, Cplex est plus performant que nos stratégies même si l'écart n'est pas très grand. Dans d'autres cas, nos stratégies sont plus performantes et, dans ce cas, l'écart est significatif. Afin de voir plus clair, nous avons séparé les instances de densité inférieure ou égale à 16% des autres. Les courbes de profils de performance des instances de densité supérieure à 16% sont présentées à la figure 4.2. Du graphique, nous constatons que notre algorithme (stratégie avec les deux types de plans coupants) est beaucoup plus performant que Cplex pour ce type d'instances. En contrepartie, l'analyse de profils de performance des instances de densité inférieure ou égale à 16% (voir la figure 4.3) montre que Cplex est plus performant que notre algorithme. Nous constatons donc que notre algorithme est efficace pour des instances de densité suprérieure à 16% (voir la figure 4.2).

4.6.2 Comparaison des valeurs de GAP à la racine

Afin de mesurer l'écart entre la borne inférieure et la valeur de la solution optimale, nous calculons le « GAP » qui est donné par la relation suivante :

$$\text{GAP} = \frac{\text{valeur de la solution optimale - valeur de la borne inférieure}}{\text{valeur de la solution optimale}} \times 100$$

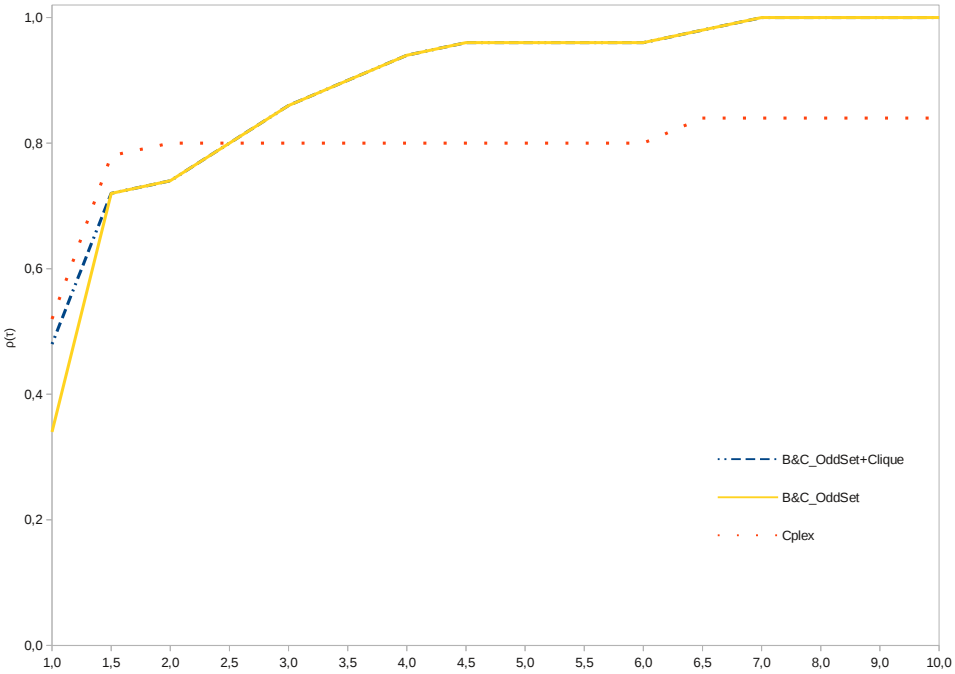

Figure 4.1 Courbes de profils de performance

Les résultats de cette comparaison sont présentés dans le tableau 4.2 ($3^{\text{ème}}$ et $9^{\text{ème}}$ colonnes) et aussi à la figure 4.4. Notons que nous n'avons considéré que les résultats des 55 instances aléatoires. Nous remarquons que dans 66% des cas, notre algorithme réussit à réduire la valeur du GAP à la racine. De plus, dans 15% des cas, la réduction est au moins égale à 35%. Il est important de préciser que pour 15 des 55 instances les procédures de séparation d'inégalités valides n'ont identifié aucune inégalité violée. En contrepartie, pour les trois instances de Hoffman et Padberg (1993), les valeurs du GAP obtenus par notre algorithme sont inférieures à celles obtenues par Cplex.

4.6.3 Quelques statistiques

Nous présentons dans le tableau 4.2 les résultats des expériences effectuées à l'aide des trois instances de Hoffman et Padberg (1993) et d'autres instances aléatoires. Les colonnes du tableau indiquent dans l'ordre de gauche à droite : le nom de l'instance, de la deuxième à la huitième colonne nous avons les résultats obtenus en utilisant l'algorithme 4.4, nous présentons : le pourcentage de variables fixées à la racine, la valeur du GAP (en %) à la racine, le nombre d'inégalités de type II, le nombre d'inégalités de clique, le nombre total d'inégalités ajoutées, le nombre de noeuds de l'arbre de branchement et le temps CPU total en secondes consommé pour la résolution du problème. Ensuite, nous présentons dans les

Tableau 4.2 Comparaison entre Cplex et l'algorithme 4.4

	Branch-and-Cut							Cplex		
Instance	Var Fix %	GAP racine %	Nb d'inégalités			Nb noeuds	Temps CPU total (s)	GAP racine %	Nb noeuds	Temps CPU total (s)
			TypeII	Clique	Total					
sppaa01	36.88	0.97	45	11	56	294	342.7	1.07	116	22.41
sppnw04	90.66	3.46	1	2	3	832	88.9	3.89	635	194.13
sppus01	99.80	0.02	13	7	20	1	572.0	1.26	5	785.56
Pb0	0	29.53	33.2	0	33.2	35137265	162943.84	30.25	17780603	126232.78
Pb1	0	34.68	18.8	9.4	28.2	214533.6	820.10	37.71	221646.6	573.53
Pb2	0	51.00	14.8	11	25.8	180287.8	645.66	54.34	142468	298.32
Pb3	0	66.77	0	0	0	1	2.11	66.77	1	2.18
Pb4	0	98.97	0	0	0	540096.8	171592.03	98.97	540096.8	171534.03
Pb5	0	88.53	0	0	0	275603.2	56788.48	88.53	275603.2	56728.48
Pb6	0	79.54	4.4	0	4.4	298332	68753.76	99.82	1045886.8	171574.66
Pb7	0	97.08	42.4	0	42.4	10624.6	4840.93	99.78	20834.2	6800.37
Pb8	0	85.13	33	0	33	9810.6	2194.81	99.76	19106.8	3433.51
Pb9	0	89.00	2.8	0	2.8	1	15.96	99.75	3776.6	297.49
Pb10	0	67.59	18.2	0	18.2	203.4	402.28	99.63	1309.6	760.89

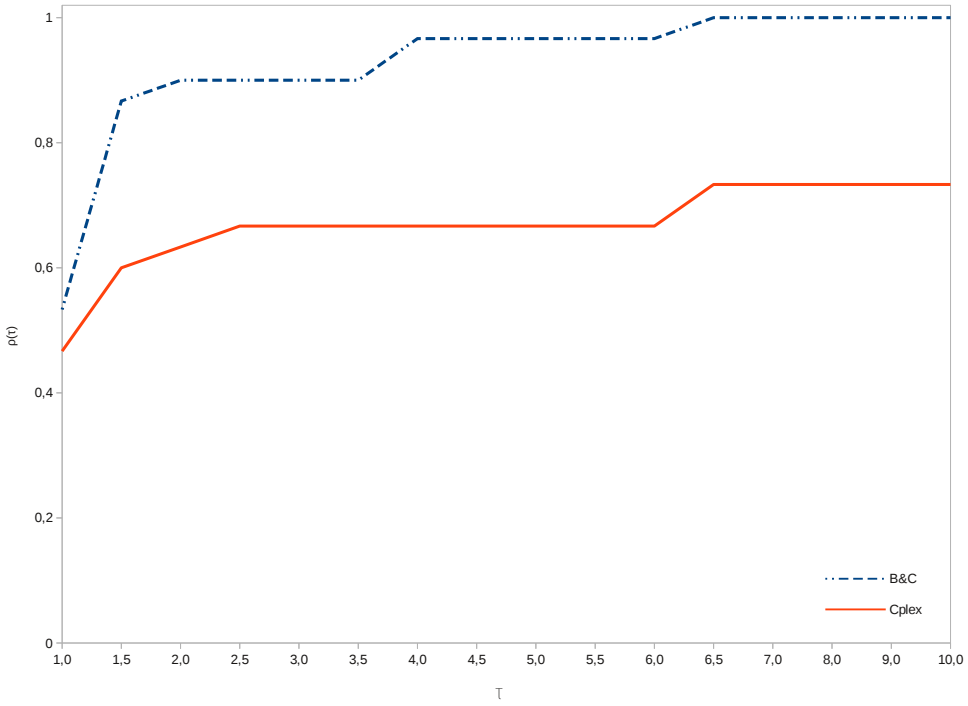

Figure 4.2 Courbes de profils des instances de densité $> 16\%$

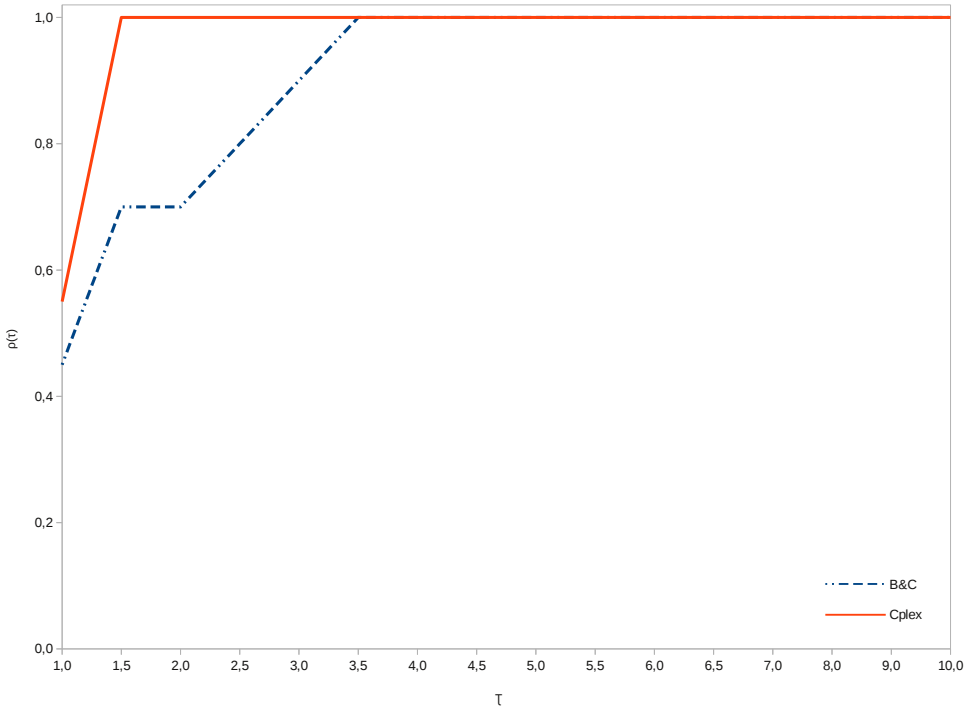

Figure 4.3 Courbes de profils des instances de densité $\leq 16\%$

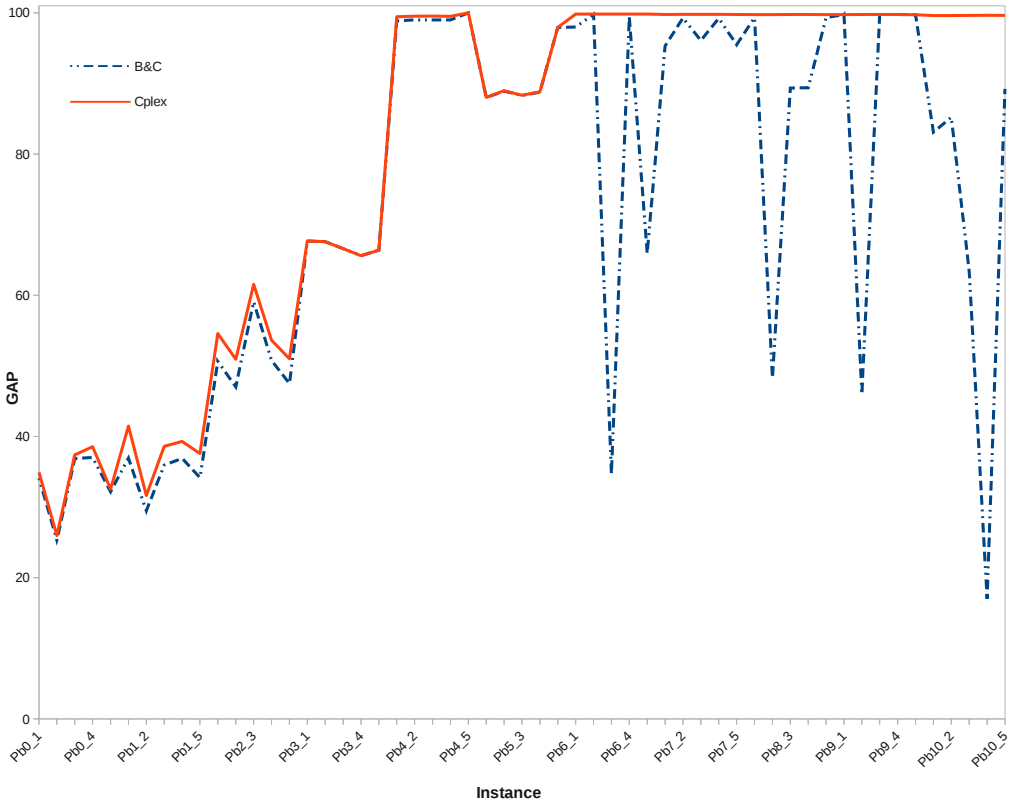

Figure 4.4 Comparaison des valeurs des GAP à la racine

trois dernières colonnes du tableau les résultats des tests obtenus en résolvant les problèmes avec Cplex par défaut. Nous présentons la valeur du GAP, le nombre de noeuds de l'arbre de branchement et le temps CPU total en secondes.

Les tests effectués à l'aide des trois instances de Hoffman et Padberg (1993) montrent que notre algorithme a éliminé plus de 90% des variables au noeud racine, sauf pour l'instance sppaa01, le pourcentage des variables éliminées est de 37%. En moyenne, 26 inégalités valides ont été ajoutées à la racine. Nous remarquons que, dans 2 cas sur 3, notre algorithme a consommé moins de temps CPU que Cplex pour résoudre les problèmes. Les réductions des temps CPU sont respectivement de 54.2% et 27.2% pour les instances sppnw04 et sppus01. Par contre, pour l'instance sppaa01, nous avons enregistré une augmentation de 93.4%. La principale différence entre les trois problèmes est la densité moyenne des colonnes. Nous constatons que la densité de sppaa01 est 1% alors que, pour les instances sppnw04 et sppus01, les densités sont respectivement 20.2% et 9.1%.

En analysant les résultats des tests effectués à l'aide des instances aléatoires, nous remarquons que nous n'avons supprimé aucune variable à la racine, contrairement aux instances de Hoffman et Padberg (1993). Ceci est dû au fait qu'à la racine l'écart entre la valeur de la

relaxation linéaire et la valeur de la borne supérieure déterminée par Cplex est très grand. Par conséquent, il est préférable d'utiliser une heuristique pour déterminer une bonne solution heuristique afin d'éliminer des variables à la racine. Nous observons aussi que pour 3 des 14 instances, nos procédures de séparation n'ont identifié aucune inégalité valide. Pour les 11 autres instances, le nombre moyen d'inégalités ajoutées est de 24.3. Environ 87% de ces dernières sont des inégalités de type II et 13% sont des inégalités de clique. Nous avons constaté qu'environ $\frac{3}{4}$ du temps total est consommé par les procédures de séparation. Ceci explique pourquoi, dans 7 cas parmi 14, Cplex est plus rapide que notre algorithme. Nous remarquons qu'en moyenne lorsque nos procédures de séparation identifient un nombre plus au moins élevé d'inégalités valides le temps total de résolution est plus important que celui enregistré par Cplex.

4.7 Conclusion

Dans le présent chapitre, nous avons proposé de résoudre le problème de partitionnement à l'aide de la méthode de séparation et évaluation progressive combinée avec les méthodes d'élimination de variables et génération d'inégalités valides. En premier, nous avons introduit deux nouvelles classes d'inégalités valides propres au SPP. Ensuite, nous avons proposé une méthode de séparation pour chaque classe. Nous avons proposé aussi une méthode de séparation des inégalités de clique. Chaque méthode de séparation est basée sur la résolution d'un problème auxiliaire en nombres entiers. Nous avons montré que, sous certaines conditions, les solutions réalisables des problèmes auxiliaires peuvent fournir des inégalités violées par la solution fractionnaire courante.

Nous avons utilisé des instances de Hoffman et Padberg (1993) et d'autres générées aléatoirement à l'aide du générateur proposé par Lewis et *al.* (2008) pour tester numériquement l'algorithme 4.4. Les résultats numériques montrent que notre algorithme est plus performant que CPLEX par défaut si la densité de la matrice des coefficients des contraintes est supérieure à 16%. En revanche, notre algorithme améliore toujours la valeur de la borne inférieure, mais est seulement plus rapide que CPLEX par défaut dans 50% des cas.

Nous constatons que les résultats sont très encourageants. Ils nous permettent de conclure que les nouvelles inégalités valides sont efficaces, mais la résolution des problèmes auxiliaires prend beaucoup de temps. Nous déduisons qu'il serait préférable de proposer une heuristique pour séparer les nouvelles inégalités valides.

CHAPITRE 5

INÉGALITÉS VALIDES ET ALGORITHMES DE SÉPARATION POUR LE MINFVS

5.1 Introduction

Dans le présent chapitre, nous nous intéressons à un problème de linguistique qui consiste à déterminer le nombre minimum de mots à connaître pour comprendre tous les mots d'un dictionnaire donné. Un dictionnaire D est un ensemble de définitions, où la définition du mot w est une séquence de mots $(w_1, w_2, \cdots, w_n)$. Si nous faisons abstraction de l'ordre des mots dans la définition du mot w, nous pouvons considérer que les mots $w_1, w_2, \cdots, w_n$ définissent le mot w. Blondin *et al.* (2008) ont formulé ce problème comme un problème du transversal de circuits de cardinalité minimale (MINFVS). Ce problème consiste à déterminer le plus petit sous-ensemble de sommets U d'un graphe orienté donné tel que si nous supprimons les sommets de U le graphe devient acyclique.

Notre objectif est d'améliorer les temps de résolution des petits dictionnaires et surtout de déterminer le transversal de circuits de cardinalité minimale pour de plus grands dictionnaires tels que Merriam Webster et WordNet. Notons que, jusqu'à la rédaction de ces lignes, aucun logiciel commercial n'a permis de déterminer la solution optimale pour chacun des deux dictionnaires. Afin de réaliser notre objectif, nous proposons de résoudre le problème à l'aide d'un algorithme combinant les méthodes de séparation et évaluation progressive, génération de contraintes et génération de plans coupants.

Dans la section 5.2, nous présentons la description et la formulation mathématique du problème. Dans la section 5.3, nous proposons d'ajouter deux classes particulières d'inégalités de Chvátal-Gomory et des inégalités de clique. Pour la séparation de chaque classe d'inégalités, nous décrivons une méthode de séparation basée sur la résolution d'un problème auxiliaire en nombres entiers. Dans la section 5.4, nous présentons les détails de l'algorithme de résolution du problème. Dans la section 5.5, nous analysons les résultats numériques des tests effectués à l'aide d'exemplaires extraits de dictionnaires de la langue anglaise qui sont disponibles sur le World Wide Web. Nous terminons par une conclusion dans la section 5.6.

5.2 Description et formulation du problème

Nous associons au dictionnaire D un graphe orienté $G_D = (V_D, A_D)$, où V_D représente l'ensemble des sommets et $A_D \subseteq V_D \times V_D$ l'ensemble des arcs. Chaque sommet $i \in V_D$

correspond à un mot du dictionnaire D. Un arc (i, j) appartient à A_D si le mot représenté par le sommet i apparaît dans la définition du mot représenté par le sommet j. Le graphe orienté $G_D = (V_D, A_D)$ est nommé graphe du dictionnaire D.

Nous avons remarqué que la plus grande composante fortement connexe de G_D est la composante source, nommé le cœur du dictionnaire, c'est-à-dire, pour chaque sommet w à l'extérieur du cœur, il existe un chemin du cœur au sommet w. Notons qu'une composante fortement connexe C_f de G_D est un sous-graphe induit maximal, où pour toute paire de sommets i et j appartenant à C_f, il existe un chemin de i vers j et il existe un chemin de j vers i (pour plus de détails voir Ahuja *et al.* (1993)). Puisque n'importe quel circuit est inclus dans l'ensemble des sommets d'une des composantes fortement connexes, alors le transversal de circuits de cardinalité minimale de G_D peut être obtenu en considérant la réunion des transversaux de circuits de cardinalité minimale de toutes les composantes fortement connexes de G_D. Dans le graphe G_D, toute composante différente du cœur est très petite par rapport à ce dernier et le transversal de circuits de cardinalité minimale de cette composante peut être déterminé facilement. Par contre, le transversal de circuits de cardinalité minimale du cœur est beaucoup plus difficile à déterminer si nous utilisons les méthodes combinatoires habituelles.

Notre objectif est de déterminer le transversal de circuits de cardinalité minimale du cœur d'un dictionnaire donné en utilisant un algorithme basé sur la génération de contraintes et de plans coupants. Mais comme notre algorithme consomme beaucoup de temps, nous réduisons le cœur le plus possible en appliquant des techniques de réduction décrites par Lin et Jou (2000).

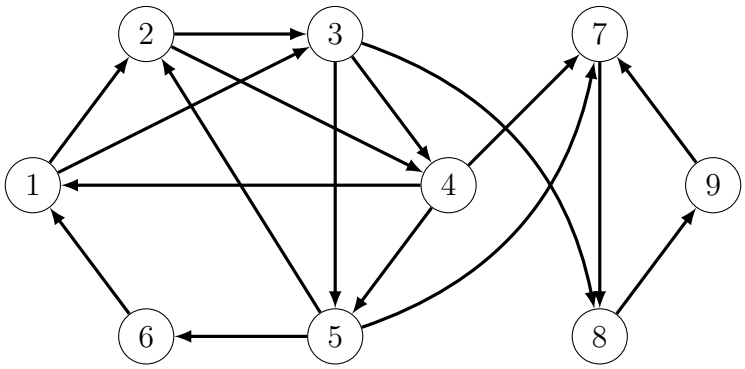

Figure 5.1 Exemple de graphe de dictionnaire

La figure 5.1 représente le graphe d'un dictionnaire. Nous remarquons qu'il existe deux composantes fortement connexes $\{1, 2, 3, 4, 5, 6\}$ et $\{7, 8, 9\}$. Il est très facile de déterminer le transversal de circuits de cardinalité minimale de la composante $\{7, 8, 9\}$. Le cœur du

dictionnaire est composé des sommets 1, 2, 3, 4, 5 et 6. Nous remarquons aussi que le sommet 6 ne possède qu'un seul arc entrant et un seul arc sortant, nous pouvons contracter l'arc $(6, 1)$ en un sommet 1^*. Le cœur devient donc $\{1^*, 2, 3, 4, 5\}$. Cette opération n'influence pas la cardinalité du transversal de circuits du cœur. Nous constatons que le transversal de circuits de cardinalité minimale de G_D est égal à la réunion des transversaux de circuits de cardinalité minimale des deux composantes fortement connexes.

Nous associons au cœur réduit du dictionnaire un graphe orienté $G = (V, A)$. Soit U un sous-ensemble de V et soit la variable binaire x_j qui prend la valeur 1 si et seulement si le sommet j appartient au sous-ensemble U et 0 sinon. Soit $\mathcal{C}$ une famille de circuits sans corde de G. Soit le coefficient a_{ij} qui prend la valeur 1 si et seulement si le sommet j appartient au circuit i et 0 sinon. Le sous-ensemble U de V est un transversal de circuits si et seulement si la contrainte $\sum_{j \in V} a_{ij} x_j \geq 1$ est vérifiée pour tout circuit i de $\mathcal{C}$. Le problème de transversal de circuits de cardinalité minimale se formule comme un programme linéaire en nombres entiers.

$$(IP_{MFVS}) \begin{cases} \min \quad \sum_{j \in V} x_j & & (5.1) \\ \quad \sum_{j \in V} a_{ij} x_j \geq 1 & \forall i \in \mathcal{C} & (5.2) \\ \quad x_j \in \{0, 1\} & \forall j \in V & (5.3) \end{cases}$$

L'objectif (5.1) représente le nombre total de sommets appartenant au transversal de circuits. Les contraintes (5.2) assurent que chaque circuit est couvert par au moins un sommet appartenant au transversal de circuits. Les contraintes (5.3) indiquent que les variables x_j sont binaires.

Il est important de noter que le nombre de contraintes du problème (IP_{MFVS}) est très élevé : il croît de façon exponentielle selon la taille du graphe G (c'est-à-dire $|A|$). Il est aussi possible de formuler le problème comme un problème de multi-flot (voir Even *et al.* 1998), mais la valeur de la relaxation linéaire ne fournit pas une bonne borne inférieure pour le problème en nombres entiers. Dans la section 5.4, nous proposons la résolution du problème par une méthode basée sur la génération de contraintes.

5.3 Les inégalités valides pour le MINFVS

Dans le but de réduire le domaine réalisable de la relaxation linéaire de (IP_{MFVS}), nous avons choisi d'ajouter deux types d'inégalités valides qui sont des cas particuliers des inégalités valides de Chvátal-Gomory de rang 1 et nous ajoutons aussi les inégalités de clique de cardinalité 3. Nos choix ne sont pas aléatoires, les inégalités valides que nous séparons correspondent

à des structures particulières dans le graphe du cœur réduit du dictionnaire $G = (V, A)$. Soit $\bar{X} = (\bar{x}_j)_{j \in V}$ une solution fractionnaire de la relaxation linéaire de (IP_{MFVS}). Dans ce qui suit, nous affectons à chaque sommet $j \in V$ le poids $\bar{x}_j$ et soit $V' = \{j \in V \mid 0 < \bar{x}_j < 1\}$. Précisons aussi que dans cette section nous ne considérons que les circuits qui sont couverts par des sommets de poids strictement inférieur à 1.

5.3.1 Les inégalités d'ensemble impair

La première classe d'inégalités que nous proposons d'ajouter est basée sur les inégalités $\{0, \frac{1}{2}\}$-Chvátal-Gomory (pour plus de détails voir Caprara et Fischetti (1996)). Afin d'éclaircir le principe, nous débutons par un exemple. Dans l'exemple de la figure 5.2, nous avons trois circuits : $C_1 = (1, 6, 2)$, $C_2 = (1, 4, 5)$ et $C_3 = (2, 4, 3, 6)$. Nous remarquons qu'il existe au moins un sommet en commun entre toute paire de ces circuits. De plus l'ensemble des sommets en commun $\{1, 2, 4\}$ constitue un cycle. Pour couvrir les trois circuits, nous avons besoin d'au moins deux sommets, c'est-à-dire que $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 2$ est valide. Nous pouvons généraliser cette observation pour tout ensemble impair de circuits.

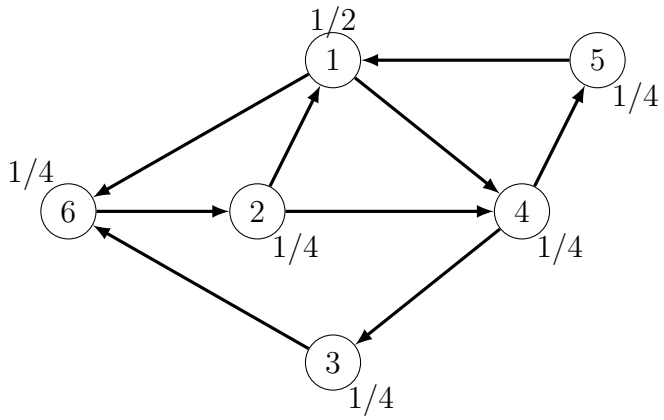

Figure 5.2 Ensemble impair de circuits

Proposition 5.3.1 *Soit $\mathcal{D}'$ un sous-ensemble impair de circuits. Soit $U' = U \cup U_0$ l'ensemble des sommets appartenant à la réunion des circuits de $\mathcal{D}'$, où chaque sommet de U appartient à un ou deux circuits de $\mathcal{D}'$ et chaque sommet de U_0 appartient à au moins trois circuits de*

$\mathcal{D}'$. Alors, l'inégalité

$$\sum_{j\in U_0}\left\lceil\frac{\sum_{i\in\mathcal{D}'}a_{ij}}{2}\right\rceil x_j+\sum_{j\in U}x_j\geq\left\lceil\frac{|\mathcal{D}'|}{2}\right\rceil \tag{5.4}$$

est valide pour (IP_{MFVS}).

Démonstration.

Des contraintes (5.2), on déduit

$$\Rightarrow \quad \sum_{i\in\mathcal{D}'}\sum_{j\in V}a_{ij}x_j \geq |\mathcal{D}'| \tag{5.5}$$

$$\Rightarrow \quad \sum_{j\in U\cup U_0}\sum_{i\in\mathcal{D}'}a_{ij}x_j \geq |\mathcal{D}'| \tag{5.6}$$

$$\Rightarrow \quad \sum_{j\in U\cup U_0}\frac{\sum_{i\in\mathcal{D}'}a_{ij}}{2}x_j \geq \frac{|\mathcal{D}'|}{2} \tag{5.7}$$

Par conséquent,

$$\sum_{j\in U\cup U_0}\left\lceil\frac{\sum_{i\in\mathcal{D}'}a_{ij}}{2}\right\rceil x_j \geq\left\lceil\frac{|\mathcal{D}'|}{2}\right\rceil \tag{5.8}$$

est valide pour (IP_{MFVS}).

Etant donné que chaque sommet $j\in U$ appartient à au plus deux circuits du sous-ensemble $\mathcal{D}'$, alors $\left\lceil\frac{\sum_{i\in\mathcal{D}'}a_{ij}}{2}\right\rceil=1$ pour tout sommet $j\in U$. Donc, l'inégalité

$$\sum_{j\in U_0}\left\lceil\frac{\sum_{i\in\mathcal{D}'}a_{ij}}{2}\right\rceil x_j+\sum_{j\in U}x_j\geq\left\lceil\frac{|\mathcal{D}'|}{2}\right\rceil$$

est valide pour (IP_{MFVS}). □

Nous avons opté pour la séparation des inégalités 5.4 telles que $U_0 = \emptyset$. Par conséquent l'inégalité 5.4 devient :

$$\sum_{j \in U} x_j \geq \left\lceil \frac{|\mathcal{D}'|}{2} \right\rceil . \tag{5.9}$$

La méthode de séparation des inégalités valides (5.9) est basée sur le même principe que pour la séparation des inégalités de coupe impaire pour le MDVSP (voir la section 3.5).

Notre objectif est de séparer un sous-ensemble $\mathcal{D}' \subset \mathcal{C}$ de circuits de cardinalité impaire, $2k+1$ où $k \in \mathbb{N}^*$, qui maximise la violation de l'inégalité valide. Nous associons à chaque circuit $i \in \mathcal{C}$ une variable binaire w_i, qui prend la valeur 1 si le circuit i appartient à l'ensemble $\mathcal{D}'$ et 0 sinon. À chaque variable $j \in V'$ nous associons deux variables binaires y_j et z_j. La variable y_j prend la valeur 1 si le sommet j appartient à au moins un circuit sélectionné et 0 sinon. La variable z_j prend la valeur 1 si le sommet j appartient à exactement un seul circuit sélectionné, et 0 sinon.

Le problème de séparation des inégalités (5.9) se formule comme un programme linéaire en nombres entiers.

$$(PbEnsImp) \begin{cases} \min & \displaystyle\sum_{j \in V'} \bar{x}_j y_j - k & & (5.10) \\ \text{s.c.} & \displaystyle\sum_{i \in \mathcal{C}} w_i = 2k+1 & & (5.11) \\ & \displaystyle\sum_{i \in \mathcal{C}} a_{ij} w_i - 2y_j + z_j = 0 & \forall j \in V' & (5.12) \\ & w_i \in \{0,1\} & \forall i \in \mathcal{C} & (5.13) \\ & y_j \in \{0,1\} & \forall j \in V' & (5.14) \\ & z_j \in \{0,1\} & \forall j \in V' & (5.15) \\ & k_{min} \leq k \leq k_{max} & k \in \mathbb{N} & (5.16) \end{cases}$$

L'objectif (5.10) représente la différence entre le poids total des sommets appartenant à la réunion des circuits sélectionnés et la valeur de la variable k. La contrainte (5.11) impose que la cardinalité du sous-ensemble $\mathcal{D}'$ sélectionné soit impaire. Les contraintes (5.12) assurent que chaque sommet appartient à au plus deux circuits de $\mathcal{D}'$. Les contraintes (5.13), (5.14) et (5.15) précisent que les variables w_i, y_j et z_j sont binaires. La contrainte (5.16) définit le domaine de la variable k. Nous imposons que k varie entre les paramètres k_{min} et k_{max} pour réduire le temps de résolution du problème auxiliaire ($PbEnsImp$).

Proposition 5.3.2 *Toute solution réalisable du problème* ($PbEnsImp$) *de valeur stricte-*

ment inférieure à 1 *induit une inégalité valide violée définie par la proposition 5.3.1.*

Démonstration. Soit (w, y, z, k) une solution réalisable du problème $(PbEnsImp)$ qui définit les ensembles suivants

- $\mathcal{D}' = \{i \in \mathcal{C} \mid w_i = 1\}$
- $U = \{j \in V' \mid y_j = 1\}$

La valeur de la solution réalisable est $\sum_{j \in V'} \bar{x}_j y_j - k$.

Par conséquent,

- par la contrainte (5.11), $\sum_{i \in \mathcal{C}} w_i = \sum_{i \in \mathcal{D}'} w_i = 2k + 1 = |\mathcal{D}'|$, donc l'ensemble $\mathcal{D}'$ contient tous les circuits sélectionnés, de plus, sa cardinalité est impaire,
- par les contraintes (5.12), $\sum_{i \in \mathcal{D}'} a_{ij} w_i = 2 - z_j$, et $z_j \in \{0, 1\}$, $\forall j \in U$, donc tous les sommets du sous-ensemble U appartiennent à au plus deux circuits de $\mathcal{D}'$,
- il n'existe pas de sommet appartenant à au moins trois circuits de $\mathcal{D}'$, donc $U_0 = \emptyset$,
- il s'ensuit que la valeur de cette solution est $\sum_{j \in U} \bar{x}_j - \dfrac{|\mathcal{D}'| - 1}{2}$.

Nous avons donc séparé un sous-ensemble de circuits $\mathcal{D}'$ de cardinalité impaire, où chaque sommet de U appartient à au plus deux circuits de $\mathcal{D}'$. De plus, si $\sum_{j \in U} \bar{x}_j - \dfrac{|\mathcal{D}'| - 1}{2} < 1$, alors nous avons identifié une inégalité violée par la solution fractionnaire de (IP_{MFVS}). □

Pour la séparation des inégalités d'ensemble impair, nous n'avons considéré que les sommets de poids fractionnaire. Nous devons ajouter les sommets de poids nul appartenant à un ou deux circuits de $\mathcal{D}'$ à l'ensemble U et les sommets appartenant à au moins trois circuits de $\mathcal{D}'$ à l'ensemble U_0. Par conséquent l'inégalité 5.4 est aussi violée par la solution fractionnaire.

L'algorithme 5.1 détaille la procédure de séparation des inégalités (5.4). Pour toute solution fractionnaire de la relaxation linéaire de (IP_{MFVS}), nous résolvons le problème auxiliaire $(PbEnsImp)$. Pour toute solution réalisable trouvée de valeur strictement inférieure à 1 de ce dernier, nous retirons les ensembles $\mathcal{D}'$ et U. Nous complétons ensuite l'inégalité en ajoutant les variables de valeur nulle, c'est-à-dire, nous considérons tous les sommets de poids nul appartenant à au moins un circuit de $\mathcal{D}'$. Enfin, nous ajoutons l'inégalité valide correspondante à la liste des inégalités valides. La procédure de séparation des inégalités d'ensemble impair est décrite par l'algorithme 5.1.

5.3.2 Les inégalités de $\{0, \frac{1}{k-1}\}$-Chvátal-Gomory

La seconde classe d'inégalités que nous proposons d'ajouter est aussi un cas particulier des inégalités de Chvátal-Gomory de rang 1, où le second membre de l'inégalité est égal à

Algorithme 5.1 Séparation d'inégalités d'ensemble impair $(Solution, ListInégValid)$

extraire la solution fractionnaire du MINFVS
résoudre le problème $(PbEnsImp)$
pour toute solution réalisable de valeur < 1 **faire**
  extraire la valeur de k
  extraire $\mathcal{D}'$ l'ensemble des circuits $i \in \mathcal{C}$ tels que $w_i = 1$,
  extraire U l'ensemble des sommets $j \in V'$ tels que $y_j = 1$,
  $U_0 := \emptyset$
  pour chaque sommet $j \in V \backslash U$ tel que $x_j = 0$ **faire**
    si $1 \leq \sum_{i \in \mathcal{D}'} a_{ij} \leq 2$ **alors**
      $U := U \cup \{j\}$
    finsi
    si $\sum_{i \in \mathcal{D}'} a_{ij} \geq 3$ **alors**
      $U_0 := U_0 \cup \{j\}$
      $l_j := \left\lceil \frac{\sum_{i \in \mathcal{D}'} a_{ij}}{2} \right\rceil$
    finsi
  finpour
  ajouter l'inégalité $\sum_{j \in U_0} l_j x_j + \sum_{j \in U} x_j \geq \left\lceil \frac{|\mathcal{D}'|}{2} \right\rceil$ à $ListInégValid$
finpour

$\left\lceil \frac{k}{k-1} \right\rceil = 2$, pour tout $k \geq 3$ (d'autres cas plus généraux sont présentés par Caprara *et al.* 2000).

Dans la figure 5.3, nous avons quatre circuits sans corde $C_1 = (1, 6, 2, 4)$, $C_2 = (1, 6, 3, 4)$, $C_3 = (2, 3, 5)$ et $C_4 = (2, 4, 5)$. Nous remarquons que les sommets 2 et 4 appartiennent à trois des quatre circuits, chacun des sommets 3, 5 et 6 fait partie de deux circuits et le sommet 1 appartient à un seul circuit. Pour couvrir l'ensemble des circuits, nous avons besoin d'au moins 2 sommets, alors que $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = \frac{7}{4}$. Donc, l'inégalité $\sum_{j=1}^{6} x_j \geq 2$ est valide pour (IP_{MFVS}).

Proposition 5.3.3 *Soit $\mathcal{D}'$ un sous-ensemble de circuits. Soit U l'ensemble des sommets appartenant à la réunion des circuits de $\mathcal{D}'$. Alors, l'inégalité*

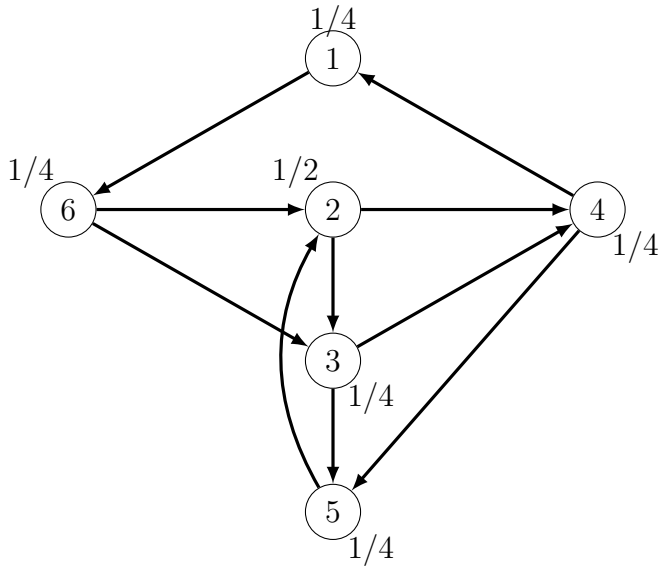

Figure 5.3 Exemple de structure correspondant aux inégalités de $\{0, \frac{1}{k-1}\}$-Chvátal-Gomory

$$\sum_{j \in U} \left\lceil \frac{\sum_{i \in \mathcal{D}'} a_{ij}}{|\mathcal{D}'| - 1} \right\rceil x_j \geq 2 \tag{5.17}$$

est valide pour (IP_{MFVS}).

Démonstration. Des contraintes (5.2), on déduit

$$\Rightarrow \quad \sum_{i \in \mathcal{D}'} \sum_{j \in V} a_{ij} x_j \geq |\mathcal{D}'| \tag{5.18}$$

$$\Rightarrow \quad \sum_{j \in U} \sum_{i \in \mathcal{D}'} a_{ij} x_j \geq |\mathcal{D}'| \tag{5.19}$$

$$\Rightarrow \quad \sum_{j \in U} \frac{\sum_{i \in \mathcal{D}'} a_{ij}}{|\mathcal{D}'| - 1} x_j \geq \frac{|\mathcal{D}'|}{|\mathcal{D}'| - 1} \tag{5.20}$$

Par conséquent,

$$\sum_{j\in U} \left\lceil \frac{\sum_{i\in D'} a_{ij}}{|\mathcal{D}'|-1} \right\rceil x_j \geq \left\lceil \frac{|\mathcal{D}'|}{|\mathcal{D}'|-1} \right\rceil \tag{5.21}$$

$$\Rightarrow \quad \sum_{j\in U} \left\lceil \frac{\sum_{i\in \mathcal{D}'} a_{ij}}{|\mathcal{D}'|-1} \right\rceil x_j \geq 2 \tag{5.22}$$

est valide pour (IP_{MFVS}).

□

Nous cherchons à identifier un sous-ensemble de circuits $\mathcal{D}' \subset \mathcal{C}$, qui minimise la valeur $\sum_{j\in U} \left\lceil \frac{\sum_{i\in \mathcal{D}'} a_{ij}}{|\mathcal{D}'|-1} \right\rceil \bar{x}_j$. Nous associons à chaque circuit $i \in \mathcal{C}$ une variable binaire w_i, qui prend la valeur 1 si le circuit i appartient à l'ensemble $\mathcal{D}'$ et 0 sinon. À chaque sommet $j \in V'$, nous associons une variable binaire y_j et une variable entière z_j. La variable y_j prend la valeur 1 si le sommet j appartient à au moins un des circuits de D' et 0 sinon. La variable z_j est égale à $\left\lceil \frac{\sum_{i\in \mathcal{D}'} a_{ij}}{|\mathcal{D}'|-1} \right\rceil$. La variable z_j prend la valeur 0 si le sommet j n'appartient à aucun circuit de $\mathcal{D}'$ (c'est-à-dire $\sum_{i\in \mathcal{D}'} a_{ij} = 0$), prend la valeur 2 si le sommet j appartient à tous les circuits de $\mathcal{D}'$ (c'est-à-dire $\sum_{i\in \mathcal{D}'} a_{ij} = |\mathcal{D}'|$) et prend la valeur 1 sinon (c'est-à-dire $1 \leq \sum_{i\in \mathcal{D}'} a_{ij} \leq |\mathcal{D}'|-1$).

Le problème auxiliaire de séparation des inégalités valides (5.17) se formule comme un programme linéaire en nombres entiers.

$$
\text{(PbChG)}\begin{cases}
\min & \displaystyle\sum_{j\in V'} \bar{x}_j z_j & & (5.23)\\
\text{s.c.} & \displaystyle\sum_{i\in\mathcal{C}} w_i = k & & (5.24)\\
& \displaystyle\sum_{i\in\mathcal{C}} a_{ij} w_i \leq M y_j & \forall j \in V' & (5.25)\\
& \displaystyle\sum_{i\in\mathcal{C}} a_{ij} w_i \leq z_j + k - 2 & \forall j \in V' & (5.26)\\
& z_j - y_j \geq 0 & \forall j \in V' & (5.27)\\
& w_i \in \{0,1\} & \forall i \in \mathcal{C} & (5.28)\\
& y_j \in \{0,1\}. & \forall j \in V' & (5.29)\\
& z_j \in \{0,1,2\} & \forall j \in V' & (5.30)\\
& k \in \mathbb{N}, k \geqslant 3 & & (5.31)
\end{cases}
$$

L'objectif (5.23) représente le poids total des sommets appartenant à la réunion des sommets de $\mathcal{D}'$. La contrainte (5.24) permet de calculer la cardinalité de l'ensemble $\mathcal{D}'$. Les contraintes (5.25) assurent que y_j est égale à 1 si le sommet j appartient à au moins un des circuits de $\mathcal{D}'$. Notons que M est une constante positive très grande. Les contraintes (5.26) et (5.27) déterminent la valeur de la variable z_j. Les contraintes (5.28), (5.29), (5.30) et (5.31) définissent les domaines des variables w_i, y_i, z_i et k.

Proposition 5.3.4 *Toute solution réalisable du problème (PbChG) de valeur strictement inférieure à 2 induit une inégalité valide (5.17) de (IP_{MFVS}) violée.*

Démonstration. Soit (w, y, z, k) une solution réalisable du problème ($PbChG$) qui définit les ensembles suivants

- $\mathcal{D}' = \{i \in \mathcal{C} \mid w_i = 1\}$
- $U = \{j \in V' \mid y_j = 1\}$

La valeur de la solution réalisable est $\displaystyle\sum_{j\in V'} \bar{x}_j z_j$.

Il s'ensuit que :

- la cardinalité du sous-ensemble des circuits sélectionnés $\mathcal{D}'$ est k, puisque $\displaystyle\sum_{i\in\mathcal{C}} w_i = \sum_{i\in\mathcal{D}'} w_i = |\mathcal{D}'| = k$,
- tout sommet j appartenant au sous-ensemble U fait partie d'au moins un circuit de $\mathcal{D}'$,

d'où $\forall j \in U$, $z_j = \left\lceil \dfrac{\sum_{i \in \mathcal{D}'} a_{ij}}{|\mathcal{D}'| - 1} \right\rceil \geq 1$,

- tout sommet j n'appartenant pas à U ne fait partie d'aucun circuit de $\mathcal{D}'$, d'où $y_j = z_j = 0$.

Par conséquent, la valeur de la solution est $\sum_{j \in U} \left\lceil \dfrac{\sum_{i \in \mathcal{D}'} a_{ij}}{|\mathcal{D}'| - 1} \right\rceil \bar{x}_j$. De plus, si cette dernière est strictement inférieure à 2, alors l'inégalité (5.17) est violée par la solution fractionnaire courante de (IP_{MFVS}). □

La procédure de séparation des inégalités (5.17) est similaire à la procédure de séparation des inégalités d'ensemble impair. À l'aide de la solution fractionnaire courante de la relaxation linéaire de (IP_{MFVS}), nous construisons et nous résolvons le programme auxiliaire en nombres entiers $(PbChG)$. Pour toute solution réalisable de valeur strictement inférieure à 2, nous extrayons $\mathcal{D}'$ l'ensemble des circuits sélectionnés et U l'ensemble des sommets appartenant à la réunion des circuits de $\mathcal{D}'$. Ensuite, nous ajoutons à l'ensemble U tous les sommets de poids nul et qui appartiennent à au moins un circuit de $\mathcal{D}'$. À partir des sommets du sous-ensemble U, nous écrivons l'inégalité valide et nous l'ajoutons à la liste des inégalités valides. Les étapes de la procédure de séparation sont données par l'algorithme 5.2.

5.3.3 Les inégalités de clique de cardinalité 3

Parmi les plus importantes inégalités valides pour les problèmes de partitionnement et de couverture, nous retrouvons les inégalités de clique. Rappelons qu'une clique de cardinalité k est un sous-graphe induit complet $K = (S, E(S))$, où S est un ensemble de sommets de cardinalité k et $E(S)$ est l'ensemble de toutes les arêtes possibles entre les sommets appartenant à S. Notre objectif est d'identifier à partir du graphe du cœur réduit un sous-graphe induit, où U est l'ensemble des sommets et il existe un sous-ensemble de sommets $S \subseteq U$ tel qu'il existe un et un seul circuit reliant toute paire de sommets appartenant à S et il existe au plus un sommet en commun entre toute paire de circuits du sous-graphe induit.

Dans l'exemple de la figure 5.4, nous avons un sous-graphe induit de trois sommets dans lequel toute paire de sommets forme un circuit de longueur 2 sans corde. Si nous remplaçons chaque circuit de longueur 2 par une arête, nous obtenons une clique de cardinalité 3. Pour couvrir tous les circuits du sous-graphe induit, nous avons besoin d'au moins deux sommets, or $\sum_{j \in U} x_j = 1.5$. Par conséquent, l'inégalité $\sum_{j \in U} x_j \geq 2$, où $U = S = \{1, 2, 3\}$, est valide pour (IP_{MFVS}) et coupe la solution courante.

Algorithme 5.2 Séparation d'inégalités (5.17) ($Solution, ListInégValid$)

extraire la solution fractionnaire du MINFVS
résoudre le problème ($PbChG$)
pour toute solution réalisable de valeur < 2 **faire**
  extraire la valeur de k
  extraire $\mathcal{D}'$ l'ensemble des circuits $i \in \mathcal{C}$ tels que $w_i = 1$,
  extraire U l'ensemble des sommets $j \in V'$ tels que $y_j = 1$,
  pour tout sommet $j \in U$ **faire**
    extraire la valeur de z_j
  finpour
  pour chaque sommet $j \in V \backslash U$ tel que $x_j = 0$ **faire**
    si $\sum_{i \in \mathcal{D}'} a_{ij} \neq 0$ **alors**
      $U := U \cup \{j\}$,
      $z_j := \left\lceil \frac{\sum_{i \in \mathcal{D}'} a_{ij}}{|\mathcal{D}'| - 1} \right\rceil$
    finsi
  finpour
  ajouter l'inégalité $\sum_{j \in U} z_j x_j \geq 2$ à $ListInégValid$
finpour

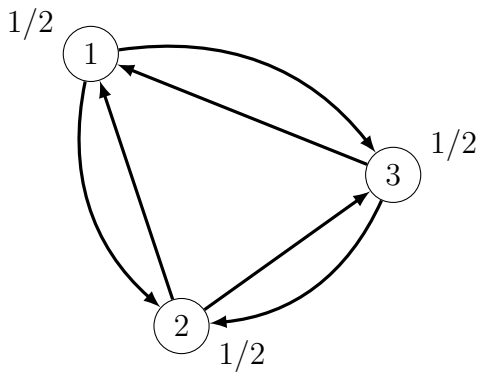

Figure 5.4 Clique de cardinalité 3

Proposition 5.3.5 *Soit $\mathcal{D}'$ un sous-ensemble de circuits sans corde. Soit U l'ensemble des sommets appartenant à la réunion des circuits de D'. Si pour toute paire de circuits de $\mathcal{D}'$, il existe un seul sommet en commun, et S est l'ensemble des sommets appartenant à la réunion*

des intersections des circuits de $\mathcal{D}'$, alors l'inégalité

$$\sum_{j \in U} x_j \geq |S| - 1 \tag{5.32}$$

est valide pour (IP_{MFVS}).

Démonstration.

Soit $\mathcal{D}'$ un sous-ensemble de circuits sans corde, U l'ensemble des sommets appartenant à la réunion des circuits de $\mathcal{D}'$ et S l'ensemble des sommets appartenant à la réunion des intersections des circuits de $\mathcal{D}'$.

Si chaque circuit appartenant à $\mathcal{D}'$ est remplacé par une arête reliant deux sommets de S, alors nous obtenons un sous-graphe complet, où le degré de chaque sommet de S est $|S|-1$ et le nombre total d'arêtes est $\frac{|S|(|S|-1)}{2}$. En d'autres termes, le sous-graphe obtenu est une clique de cardinalité $|S|$. Pour couvrir toutes les arêtes de cette dernière, nous avons besoin d'au moins $|S|-1$ sommets de S. Par conséquent, $\sum_{j \in U} x_j \geq |S|-1$ est valide pour (IP_{MFVS}).
□

La proposition 5.3.5 est vérifiée quelque soit la cardinalité du sous-ensemble S. Malheureusement, la séparation des inégalités de clique de circuits d'une cardinalité quelconque est très difficile. Nous avons élaboré un algorithme glouton pour la séparation de cliques de cardinalité k, mais nous avons constaté qu'il consomme énormément de temps. Nous avons tenté de modéliser le problème par un programme linéaire, mais nous avons constaté que la modélisation de la condition exigée entre toute paire de circuits engendre un nombre exponentiel de contraintes. Nous avons donc choisi de se limiter à des cliques de cardinalité 3, c'est-à-dire que $|S| = 3$.

Afin de séparer les inégalités définies par la proposition 5.3.5, nous cherchons à identifier un sous-ensemble de circuits $\mathcal{D}' \subset \mathcal{C}$ tel que le sous-ensemble des sommets appartenant à la réunion des circuits de $\mathcal{D}'$ maximise la violation de l'inégalité (5.32) pour $|S| = 3$. En premier, nous redéfinissons l'ensemble des sommets $V' = \{j \in V \mid 0 \leq \bar{x}_j < 1\}$. Nous associons à chaque circuit $i \in \mathcal{C}$ une variable binaire w_i, qui prend la valeur 1 si le circuit i appartient à l'ensemble $\mathcal{D}'$ et 0 sinon. À chaque variable $j \in V'$, nous associons deux variables binaires y_j et z_j. La variable y_j prend la valeur 1 si le sommet j appartient à au moins un circuit sélectionné et 0 sinon. La variable z_j prend la valeur 1 si le sommet j appartient à exactement un circuit sélectionné, et 0 sinon.

Le problème de séparation des inégalités (5.32) se formule comme un programme linéaire en nombres entiers.

$$
(\text{PbK3}) \begin{cases}
\min \quad \displaystyle\sum_{j \in V'} \bar{x}_j y_j & & (5.33) \\
\displaystyle\sum_{i \in \mathcal{C}} w_i = 3 & & (5.34) \\
\displaystyle\sum_{j \in V'} (y_j - z_j) = 3 & & (5.35) \\
\displaystyle\sum_{i \in \mathcal{C}} a_{ij} w_i - 2y_j + z_j = 0 & \forall j \in V & (5.36) \\
w_i \in \{0,1\} & \forall i \in \mathcal{C} & (5.37) \\
y_j \in \{0,1\} & \forall j \in V' & (5.38) \\
z_j \in \{0,1\} & \forall j \in V' & (5.39)
\end{cases}
$$

L'objectif (5.33) représente le poids total des sommets appartenant à la réunion des sommets de $\mathcal{D}'$. La contrainte (5.34) impose que la cardinalité de $\mathcal{D}'$ est 3. La contrainte (5.35) exige que la cardinalité de l'ensemble des sommets d'intersection des circuits est 3. Les contraintes (5.36) indiquent que chaque sommet appartient à au plus deux circuits de $\mathcal{D}'$. Les contraintes (5.37), (5.38) et (5.39) précisent que les variables w_i, y_j et z_j sont binaires.

Proposition 5.3.6 *Toute solution réalisable du problème* $(PbK3)$ *de valeur strictement inférieure à* 2 *induit une inégalité valide (5.32) du* (IP_{MFVS}) *violée.*

Démonstration. Soit (w, y, z) une solution réalisable du problème $(PbK3)$ qui définit les ensembles suivants

- $\mathcal{D}' = \{i \in \mathcal{C} \mid w_i = 1\}$
- $U = \{j \in V' \mid y_j = 1\}$
- $S = \{j \in V' \mid y_j = 1 \; et \; z_j = 0\}$

La valeur de la solution réalisable est $\displaystyle\sum_{j \in V'} \bar{x}_j y_j$.

Il s'ensuit que :

- la cardinalité de l'ensemble des circuits séléctionnés est 3, puisque $\displaystyle\sum_{i \in \mathcal{C}} w_i = \sum_{i \in \mathcal{D}'} w_i = |\mathcal{D}'| = 3$,
- tous les sommets du sous-ensemble U appartiennent à au moins un circuit de $\mathcal{D}'$; donc pour tout sommet $j \in U$, $y_j = 1$,
- tout sommet j n'appartenant pas à U est un sommet qui ne fait partie d'aucun circuit de $\mathcal{D}'$; d'où $y_j = z_j = 0$,

- tout sommet $j \in U$ appartenant à au moins deux circuits de $\mathcal{D}'$ est un sommet du sous-ensemble S, d'où $|S| = \sum_{j \in S}(y_j - z_j) = 3$.

Par conséquent, la valeur de cette solution réalisable est $\sum_{j \in U} \bar{x}_j$. Si de plus, sa valeur est strictement inférieure à $2 = |S| - 1$, alors l'inégalité (5.32) est violée par la solution fractionnaire courante. □

La procédure de séparation des inégalités de clique de circuits est similaire aux deux précédentes procédures de séparation. Nous résolvons le problème auxiliaire ($PbK3$) et pour chaque solution réalisable de valeur strictement inférieure à 2 trouvée, nous extrayons l'ensemble des circuits sélectionnés $\mathcal{D}'$, ainsi que l'ensemble des sommets appartenant à la réunion des circuits U. Nous écrivons l'inégalité de clique de cardinalité 3 et nous l'ajoutons à la liste des inégalités valides. Les détails de la procédure de séparation sont donnés par l'algorithme 5.3. Notons que nous considérons tous les sommets de poids strictement inférieur à 1.

Algorithme 5.3 Séparation d'inégalités de clique de cardinalité 3 ($Solution, ListInégValid$)

1: extraire la solution fractionnaire du MINFVS
2: résoudre le problème ($PbK3$)
3: **pour** toute solution réalisable de valeur < 2 **faire**
4: extraire $\mathcal{D}'$ l'ensemble des circuits $i \in \mathcal{C}$ tels que $w_i = 1$
5: extraire U l'ensemble des sommets $j \in V'$ tels que $y_j = 1$
6: ajouter l'inégalité $\sum_{j \in U} x_j \geq 2$ à $ListInégValid$
7: **finpour**

5.4 Méthode de résolution du MINFVS

Etant donné que le nombre de contraintes de (IP_{MFVS}) est très élevé, nous avons utilisé la génération de contraintes pour la résolution de la relaxation linéaire et pour la résolution du programme linéaire en nombres entiers. À chaque étape de notre méthode, nous considérons seulement un sous-ensemble $\mathcal{D}$ de l'ensemble des circuits $\mathcal{C}$. Dans ce qui suit, nous utiliserons la notation ($IP_{MFVS}(\mathcal{D})$) (resp. $LP_{MFVS}(\mathcal{D})$) pour faire référence au programme en nombres entiers (IP_{MFVS}) (resp. la relaxation linéaire (LP_{MFVS})) en remplaçant $\mathcal{C}$ par $\mathcal{D}$. Il est important de préciser que LP_{MFVS} est un problème réalisable.

Initialement, nous démarrons avec un sous-ensemble de circuits vide, noté $\mathcal{D}$, nous résolvons le problème ($LP_{MFVS}(\mathcal{D})$). Soit $x = (x_j)_{j \in V}$ la solution du problème. Nous associons à chaque sommet $j \in V$ le poids x_j. Pour chaque sommet $j \in V$, nous utilisons l'algorithme de Dijkstra pour déterminer le plus court chemin de j à lui-même contenant au moins un arc. Si son poids est strictement inférieur à 1, alors nous avons détecté un circuit C violé.

Si le circuit C est sans corde, alors nous l'ajoutons à l'ensemble des circuits de $\mathcal{D}$. Sinon, nous extrayons de C tous les circuits sans corde et nous les ajoutons à $\mathcal{D}$, en s'assurant qu'il n'existe pas de circuits identiques (voir l'algorithme 5.4). Nous résolvons le nouveau problème ($LP_{MFVS}(\mathcal{D})$) et nous répétons la même opération jusqu'à obtenir une solution fractionnaire qui ne viole aucun circuit de $\mathcal{C}$. La principale raison pour laquelle nous résolvons plusieurs programmes linéaires est que nous avons constaté que le sous-ensemble de circuits $\mathcal{D}$ (obtenu par cette technique) fournit une bonne borne inférieure au problème $IP_{MFVS}(\mathcal{C})$.

Dans le but d'améliorer la valeur de la borne inférieure, nous appliquons les procédures décrites par les algorithmes 5.1, 5.2 et 5.3 pour la séparation des inégalités valides en utilisant la dernière solution fractionnaire. S'il existe des inégalités violées alors nous les ajoutons au problème ($LP_{MFVS}(\mathcal{D})$) et nous résolvons à nouveau. Nous procédons de la même façon jusqu'à qu'il n'existe plus d'inégalité violée par la dernière solution fractionnaire. Ensuite, nous débutons la résolution du programme linéaire en nombres entiers ($IP_{MFVS}(\mathcal{D})$). Durant la résolution, s'il existe une solution réalisable de ($IP_{MFVS}(\mathcal{D})$), et si elle ne viole aucun circuit de $\mathcal{C}$, alors cette solution est acceptée. Sinon nous ajoutons les circuits violés à $\mathcal{D}$ et nous continuons la résolution. Les circuits violés sont obtenus à l'aide de la procédure décrite par l'algorithme 5.4.

Notons que nous utilisons CPLEX pour résoudre les programmes linéaires relaxés et les programmes linéaires en nombres entiers. Pour l'ajout des circuits non couverts par les solutions entières, nous utilisons la fonction « CPXsetlazyconstraintcallbackfunc( ) » de CPLEX. La méthode de résolution du problème est décrite par l'algorithme 5.5.

5.5 Application et résultats

Nous avons utilisé CPLEX 12.4.0.0 pour résoudre nos problèmes sur la machine Briarée du Réseau Calcul Québec de Haute Performance (RQCHP). Briarée possède 630 noeuds, chacun contient 2 processeurs Intel (Intel Xeon X5650 2.66GHz Westmere-EP). Notons que nous utilisons le branchement parallèle de CPLEX. Le maximum autorisé sur Briarée est 12 noeuds en parallèle pour une durée maximale de 3 semaines.

Nous présentons dans le tableau 5.1 les caractéristiques des cœurs réduits de quatre dictionnaires que nous avons utilisés pour tester notre algorithme et deux autres instances intermédiaires que nous avons créé à partir des graphes des cœurs réduits des dictionnaires Merriam-Webster et WordNet. À partir du graphe de chaque cœur réduit, nous avons construit un sous-graphe induit qui contient environ $\frac{2}{3}$ des sommets du graphe initial. Notons que le sous-graphe induit obtenu n'est pas forcément fortement connexe. Les nouvelles instances MW* et WN* sont extraites des cœurs réduits des dictionnaires Merriam-Webster

Algorithme 5.4 Séparation des circuits sans corde $(G = (V, A), Solution, \mathcal{D})$

Soit $x = (x_j)_{j \in V}$ une solution du problème
Soit $\mathcal{C}$ l'ensemble de tous les circuits de G
Soit $\mathcal{D}$ un sous-ensemble de circuits
pour chaque sommet $j \in V$ **faire**
  associer le poids x_j à j
finpour
pour chaque sommet $j \in V$ telle que $x_j < 1$ **faire**
  Dijkstra(G, x, j)
  si il existe un plus court chemin de j à j de longueur l et de poids strictement inférieur à 1 **alors**
    $C :=$ le plus court chemin de j à j
    si C est un circuit sans corde et $C \notin \mathcal{D}$ **alors**
      ajouter C à $\mathcal{D}$
    sinon
      extraire de C tout circuit sans corde et l'ajouter à $\mathcal{D}$
    finsi
  finsi
finpour

et WordNet, respectivement.

Nous remarquons que les graphes associés aux cœurs réduits des dictionnaires Merriam-Webster et WordNet sont plus denses (c'est-à-dire contiennent un grand nombre de sommets et d'arcs) que les graphes associés aux cœurs réduits des dictionnaires Cambridge et Longman. D'autre part, les graphes des instances MW* et WN* contiennent plus de sommets que le graphe associé au cœur réduit de Longman et environ le même nombre d'arcs.

Tableau 5.1 Caractéristiques des instances testées

Instances	Nb de sommets	Nb d'arcs	Nb moyen d'arcs sortant
Cambridge	647	6114	9.45
Longman	873	8686	8.56
Merriam-Webster	2191	14276	7.72
WordNet	2135	14713	6.89
MW*	1460	9315	6.38
WN*	1423	8099	5.69

En effectuant plusieurs tests, nous avons remarqué que la résolution du problème avec la méthode de séparation et évaluation progressive fournit un arbre de branchement avec un nombre démesuré de noeuds de branchement. Nous avons constaté qu'il existe beaucoup de symétrie dans l'arbre de branchement, c'est-à-dire qu'il existe beaucoup de solutions op-

Algorithme 5.5 Résolution de IP_{MFVS} par génération de contraintes ($G = (V, A)$)

Soit $\mathcal{C}$ l'ensemble de tous les circuits du graphe G
$\mathcal{D} \leftarrow \emptyset$
résoudre $LP_{MFVS}(\mathcal{D})$
soit x la solution optimale de $LP_{MFVS}(\mathcal{D})$
appliquer la procédure 5.4 pour déterminer les circuits sans corde de longueur ≤ 5 et de poids < 1 dans G
tant que il existe un circuit C de $\mathcal{C}$ violé par la solution x **faire**
  ajouter à $\mathcal{D}$ le circuit C
  résoudre $LP_{MFVS}(\mathcal{D})$
  soit x la solution optimale de $LP_{MFVS}(\mathcal{D})$
  appliquer la procédure 5.4 pour déterminer les circuits sans corde de poids < 1 dans G
fintant que
appliquer les procédures 5.1, 5.2 et 5.3 de séparation des inégalités valides.
tant que il existe des inégalités violées **faire**
  ajouter les inégalités valides au $LP_{MFVS}(\mathcal{D})$
  résoudre $LP_{MFVS}(\mathcal{D})$
  appliquer les procédure 5.1, 5.2 et 5.3 de séparation des inégalités valides.
fintant que
lancer la résolution de $IP_{MFVS}(\mathcal{D})$
pour toute solution réalisable trouvée x **faire**
  si la solution x couvre tous les circuits du graphe G **alors**
    accepter la solution x
  sinon
    appliquer la procédure 5.4 pour déterminer les circuits sans corde de poids < 1 dans G
    tant que il existe un circuit C de $\mathcal{C}$ violé par la solution x **faire**
      ajouter le circuit C à $\mathcal{D}$
    fintant que
  finsi
finpour
retourner la solution optimale x

timales. C'est pour cette raison que le solveur n'arrive pas à supprimer des noeuds et par conséquent il doit exploiter tous les noeuds de l'arbre. Nous avons testé deux techniques de branchement « meilleure-borne » et « en profondeur ». Avec la première technique CPLEX arrête la résolution après 8 heures par manque de mémoire et avec la deuxième technique le branchement dure des semaines sans réussir à prouver l'optimalité de la dernière solution trouvée. Sur la base de ces observations, nous avons choisi de faire un branchement en profondeur et de limiter le temps de résolution à 26 heures.

Dans le tableau 5.2, nous présentons les résultats des tests obtenus en résolvant les problèmes à l'aide de Cplex par défaut. Nous constatons qu'en moyenne, nous avons résolu 7

programmes linéaires en des temps variant entre 10 et 80 secondes. Pour les instances Cambridge, Longman et MW*, nous avons trouvé les solutions optimales en 8, 132 secondes et 2 heures 31 minutes, respectivement. Nous avons donc la preuve d'optimalité pour ces trois instances. Pour les instances Merriam-Webster, WordNet et WN*, le solveur n'est pas capable de déterminer une solution réalisable facilement ; c'est seulement après plusieurs noeuds de branchement et plusieurs tentatives qu'il réussit à trouver une solution réalisable et l'écart entre cette dernière et la borne inférieure est très grand. Notons que nous avons limité le temps de résolution total à 26 heures. Les valeurs des meilleures solutions trouvées des instances Merriam-Webster, WordNet et WN* sont, respectivement, 305, 323 et 164. Les valeurs des GAP enregistrés par ces solutions sont 10.69%, 15.06% et 5.17%, respectivement. Pour ces trois instances, CPLEX n'a pas réussi à trouver les solutions optimales.

Dans le tableau 5.3, nous présentons les résultats des tests obtenus en résolvant les problèmes à l'aide de l'algorithme 5.5. Étant donné qu'il est facile de résoudre les instances Cambridge et Longman, nous n'avons pas appliqué l'algorithme 5.5 pour les résoudre. Nous avons constaté que, dans les graphes des instances Merriam-Webster, WordNet, MW* et WN*, il existe un grand nombre d'inégalités violées par les solutions fractionnaires. Ceci nous a obligé à inclure d'autres critères d'arrêt dans la procédure de séparation des inégalités valides. Nous avons décidé que si le nombre d'inégalités valides ajoutées est supérieur ou égal à 8% du nombre de contraintes ou si l'amélioration de la valeur de la borne inférieure est inférieure à 0.1, alors nous arrêtons la procédure de séparation des inégalités valides.

Nous remarquons que nous trouvons un grand nombre d'inégalités d'ensemble impair par rapport aux inégalités de $\{0, \frac{1}{k-1}\}$-Chvátal-Gomory et aux inégalités de clique de cardinalité 3. En moyenne, 95% des inégalités valides trouvées sont des inégalités d'ensemble impair. Dans le graphe de l'instance Merriam-Webster, nous avons identifié plus d'inégalités d'ensemble impair que dans les graphes des autres instances, En revanche, nous avons identifié plus d'inégalités de $\{0, \frac{1}{k-1}\}$-Chvátal-Gomory et d'inégalités de clique dans le graphe de l'instance WN* que dans les graphes des autres instances. Nous observons que nous avons ajouté beaucoup plus d'inégalités valides aux problèmes Merriam-Webster et WN* qu'aux problèmes WordNet et MW*. Nous constatons aussi que l'amélioration de la valeur de la solution fractionnaire n'est pas très grande après l'ajout des inégalités valides. Il est possible d'augmenter la valeur de la borne inférieure en ajoutant plus d'inégalités valides si nous modifions les critères d'arrêt de la procédure de séparation, mais en contrepartie nous augmentons le temps de séparation. Nous avons trouvé les solutions optimales des instances MW* et WN* en 2 heures 17 minutes et 5 heures 16 minutes, respectivement. Rappelons qu'en 26 heures CPLEX n'a pas résussi à trouver la solution optimale de l'instances WN*. Nous enregistrons donc une réduction d'environ 80% du temps total de résolution de cette instance avec notre

algorithme. Selon les résultats présentés, l'ajout des inégalités valides permet de réduire la taille de l'arbre de branchement. D'autre part, nous avons trouvé des solutions réalisables de valeurs 296 et 311 pour les instances Merriam-Webster et WordNet, respectivement, qui sont meilleures que celles trouvées par Cplex. Notons que nous savons que ces solutions ne sont pas optimales parce que nous avons trouvé des solutions réalisables de valeurs 291 et 309 pour Merriam-Webster et WordNet, respectivement, après plusieurs jours.

5.6 Conclusion

Dans ce chapitre, nous avons étudié une des applications du problème de transversal de circuits de cardinalité minimale. Nous avons essayé de résoudre le problème de détermination du nombre minimum de mots à connaître pour comprendre tous les mots d'un dictionnaire donné. Nous avons présenté un algorithme combinant les méthodes de génération de contraintes et de plans coupants pour la résolution du problème. Nous avons proposé la séparation de deux classes particulères des inégalités de Chvátal-Gomory et des inégalités de clique par la résolution de problèmes auxiliaires en nombres entiers. Nous avons montré que certaines solutions réalisables des problèmes auxiliaires peuvent fournir des inégalités violées par la solution fractionnaire.

Nous avons utilisé des instances extraites de dictionnaires de la langue anglaise pour tester numériquement notre algorithme. Nous avons observé qu'il est très facile de résoudre le problème dont le graphe possède moins de 900 sommets et 9000 arcs, mais il devient très difficile si le graphe est plus dense. Les résultats numériques montrent clairement que l'ajout des inégalités valides permet de trouver de meilleures solutions réalisables dans le temps imparti et dans certains cas de trouver la solution optimale du problème tout en améliorant considérablement le temps total de résolution. Nous avons constaté aussi que le problème auxiliaire de séparation des inégalités d'ensemble impair a permis d'identifier un très grand nombre d'inégalités violées par les solutions fractionnaires.

Nous avons obsevé qu'il existe beaucoup de symétrie dans l'arbre de banchement. Pour réduire cette symétrie, nous avons pénalisé le coefficient de chaque variable dans la fonction objectif en fonction de sa densité, nous avons ajouté une contrainte pour borner l'ancienne fonction objectif et nous avons résolu un problème de maximisation. Malheureusement, cette technique n'a pas permis de résoudre le problème non plus. Nous pensons qu'il est préférable de gérer le branchement tout en essayant de briser la symétrie (voir Ostrowski et *al* . (2009)). Nous pensons aussi qu'il faut ajouter des inégalités valides permettant d'éliminer des cas de symétrie (voir Méndez-Dìaz et Zabala (2006), Friedman (2007), Kaibel et Pfetsch (2007)). Nous retrouvons dans la littérature des méthodes qui permettent de résoudre des problèmes

d'optimisation en considérant la symétrie (voir Sherali and Smith (2001) et Ghoniem and Sherali (2011)). Malheureusement, il nous a été impossible d'identifier une structure propre au problème parce qu'il est impossible de visualiser le graphe du cœur réduit (ni même un sous-graphe) étant donné qu'il est très dense.

Tableau 5.2 Résultats obtenus en appliquant CPLEX par défaut

	Cambridge	Longman	Merriam-Webster	WordNet	MW*	WN*
Nb.Variables	647	873	2191	2135	1460	1423
Nb.Contraintes	1520	3397	10976	13331	4303	5202
Valeur.Relaxation	128.09	194.90	271.32	272.55	139.35	154.92
Nb.LP résolus	6	7	8	8	7	7
Temps.Total.LP(sec)	9.23	11.65	72.23	80.00	18.55	22.07
Valeur.Meilleure Solution	132	203	305	323	148	164
GAP d'optimalité	0.0%	0.0%	10.69%	15.06%	0.0%	5.17%
Nb.Noeuds de branchement	66	29451	1430704	743337	1131105	5127972
Temps total de résolution	16.79 sec	143.65 sec	26 h	26 h	2 h 31 min	26 h
Preuve d'optimalité	✓	✓			✓	

Tableau 5.3 Résultats obtenus en appliquant l'algorithme 5.5

	Merriam-Webster	WordNet	MW*	WN*
Nb.Inég.Ensemble Impair	580	207	216	371
Nb.Inég.Chvátal-Gomory	18	5	23	32
Nb.Inég.Clique	3	2	1	11
Nb.Total d'Inég. Ajoutées	601	214	240	414
Valeur.Borne Inférieure	273.04	273.41	141.00	156.10
Valeur.Meilleure Solution	296	311	148	163
GAP d'optimalité	7.60%	11.12%	0.0%	0.0%
Nb.Noeuds de branchement	778645	365477	421529	604547
Temps de séparation	2 h	1 h 55 min	1 h 05 min	2 h 02 min
Temps total de résolution	26 h	26 h	2 h 17 min	5 h 16 min
Preuve d'optimalité			✓	✓

CHAPITRE 6

CONCLUSION

Durant les dernières décennies, beaucoup de travaux ont été consacrés à la détermination et la séparation de différentes classes d'inégalités valides pour les programmes linéaires en nombres entiers. Pour certains, il a été démontré que, dans le cas général, le problème de séparation est NP-complet. Pour d'autres, nous ignorons à ce jour si leur problème de séparation est NP-complet ou non.

Généralement, les méthodes de séparation proposées dans la littérature sont pour des problèmes d'optimisation pratiques, classés NP-complet, formulés comme des programmes linéaires en nombres entiers. Dans cette thèse, nous avons étudié des inégalités valides pour des problèmes de partitionnement et de couverture, nous avons proposé des algorithmes de séparation de chaque classe étudiée et nous avons analysé les points positifs et négatifs de chaque algorithme proposé.

6.1 Synthèse des travaux

Dans la première partie de cette thèse, nous avons étudié les inégalités valides et les algorithmes de séparation pour le MDVSP. À partir de la notion de sous-multigraphe épineux introduite par Hadjar *et al.* (2006), nous avons présenté une généralisation de cette notion. Nous avons montré qu'il est possible d'identifier des structures plus générales que la structure de cycle impair épineux, identifiée par les auteurs. Nous avons exploité la relation entre le MDVSP et le problème de couplage, et entre ce dernier et le problème du stable, pour identifier deux classes particulières d'inégalités valides propres au MDVSP. Nous avons nommé les inégalités de la première classe, les inégalités de cycle impair, étant donné que leur structure dans le graphe de conflit correspond à un cycle impair ; et nous avons nommé les inégalités de la deuxième classe, les inégalités de coupe impaire, étant donné que leur structure dans le graphe des chemins correspond à une coupe impaire. Nous avons proposé un algorithme polynomial pour la séparation des inégalités de cycle impair et nous avons séparé les inégalités de coupe impaire en résolvant un programme auxiliaire en nombres entiers. Les résultats numériques ont montré que la séparation des deux classes d'inégalités valides prend, en moyenne, 33% du temps total de résolution du MDVSP. Le temps de séparation des inégalités de cycle impair représente, en moyenne, 7.3% du temps total de séparation ; et en moyenne, le temps de séparation des inégalités de coupe impaire est égal à 92.6% du temps

total de séparation. Les résultats numériques ont aussi démontré l'efficacité de l'algorithme de type « branch-and-cut » que nous avons proposé.

Dans la deuxième partie de cette thèse, nous avons étudié les inégalités valides et les algorithmes de séparation pour le SPP. Nous avons proposé deux nouvelles classes d'inégalités valides propres au problème. Nous avons clarifié la relation entre les inégalités valides des deux classes. Nous avons montré que si l'inégalité valide de type I est violée par la solution fractionnaire courante, alors l'inégalité valide de type II l'est aussi. Nous avons proposé la séparation des inégalités de chaque classe par la résolution d'un problème auxiliaire en nombres entiers. Nous avons proposé d'ajouter les inégalités de clique et la résolution d'un problème auxiliaire pour les séparer. Les résultats numériques ont montré que l'ajout des inégalités valides de type II ainsi que les inégalités de clique fournit de meilleurs résultats. Les résultats ont montré aussi que l'algorithme de type « branch-and-cut » que nous proposons pour la résolution du SPP est plus efficace que CPLEX pour les instances dont la densité des colonnes est supérieure à 16%, et moins efficace pour les instances dont la densité est inférieure ou égale à 16%. Par contre, l'ajout des inégalités valides proposées fournit un meilleur GAP que celui de CPLEX, dans la plupart des cas. Ce résultat ne découle pas seulement de l'amélioration de la borne inférieure, mais dans certains cas, l'ajout des plans coupants aide Cplex à identifier une meilleure borne supérieure.

Dans la troisième partie de cette thèse, nous avons étudié un problème de linguistique qui consiste à déterminer le nombre minimum de mots à connaître pour comprendre toutes les définitions d'un dictionnaire donné. Ce problème a été modélisé comme un problème de transversal de circuits de cardinalité minimale. Nous avons proposé d'ajouter deux classes particulières d'inégalités de Chvátal-Gomory, les $\{0, \frac{1}{2}\}$-Chvátal-Gomory et les $\{0, \frac{1}{k-1}\}$-Chvátal-Gomory, et les inégalités de clique de cardinalité 3. Pour la séparation de chaque classe d'inégalités valides, nous avons résolu un problème auxiliaire en nombres entiers. Les résultats numériques ont montré que nos méthodes de séparation permettent d'identifier un grand nombre d'inégalités valides, mais en contrepartie elles consomment beaucoup de temps. Pour la résolution du problème nous avons présenté un algorithme de type « branch-and-cut ». Nous avons constaté que l'ajout des inégalités valides aide CPLEX à identifier des solutions réalisables plus rapidement et dans certains cas à trouver la solution optimale en peu de temps. Malheureusement, nous n'avons pas réussi à trouver la solution optimale des cœurs réduits des dictionnaires Meriem-Webster et WordNet. Notons que jusqu'à la rédaction de ces lignes, nous n'avons trouvé aucun moyen qui permet de trouver la solution optimale de chaque problème. Pour les cœurs réduits des dictionnaires Meriem-Webster et WordNet, nous avons essayé de résoudre les problèmes avec CPLEX et avec notre algorithme, les tests ont duré 3 semaines sans prouver l'optimalité de la dernière solution trouvée. Par contre, nos

résultats montrent que les inégalités valides proposées permettent d'obtenir de meilleures solutions réalisables dans un temps limite de 26 heures.

6.2 Améliorations futures

À partir des résultats numériques présentés dans le présent travail, nous concluons que les inégalités valides proposées réduisent le domaine réalisable de la relaxation linéaire des problèmes étudiés, aident à déterminer de meilleures solutions réalisables et dans la plupart des cas permettent de résoudre les problèmes en moins de temps. Néanmoins, la séparation des inégalités valides en résolvant des problèmes auxilaires consomme beaucoup de temps. Suite à cette observation, nous pensons que le développement d'heuristiques pour la séparation de chaque classe d'inégalités valides permettra d'améliorer considérablement les temps de séparation.

L'une des idées que nous pouvons exploiter pour la séparation des inégalités de coupe impaire pour le MDVSP, est d'utiliser l'algorithme polynomial proposé par Padberg et Rao (1982) pour la séparation des inégalités de couplage. La principale difficulté à surmonter avant d'adapter l'algorithme de Padberg et Rao (1982) au MDVSP est de trouver une façon de différencier les noeuds de conflit des noeuds intermédiaires, afin de construire des chemins réguliers.

Concernant la séparation des inégalités valides proposées pour résoudre le SPP, nous avons testé une heuristique basée sur une recherche tabou, mais cette dernière n'a permis d'identifier que des structures de très petites tailles. Il faut développer une heuristique plus sophistiquée. Les inégalités de type II nous permettent de déduire que pour couvrir un nombre impair de lignes nous avons besoin d'un nombre impair de colonnes avec un nombre impair d'éléments non nuls dans chaque colonne. Il est difficile d'écrire une inégalité valide à partir d'une telle sous-matrice, mais il est possible de brancher en considérant les colonnes de cette dernière.

L'une des idées que nous pensons exploiter pour résoudre le problème de transversal de circuits de cardinalité minimale est d'identifier une classe d'inégalités valides (de préférence des facettes) qui prend en considération la symétrie entre les sommets du graphe de dictionnaire. Ce type d'inégalités aidera le solveur à éliminer les solutions équivalentes.

RÉFÉRENCES

[1] Ahuja R.K, Magnanti T.L, Orlin J.B. Network Flows. Theory, algorithms, and applications. *Prentice Hall*, 1993.

[2] Balas E., Padberg M.W. Set Partitioning : A Survey. *SIAM Review* **18** 710–760, 1976.

[3] Bertossi, A. A., Carraresi, P., Gallo, G. On some matching problems arising in vehicle scheduling models. *Networks* **17** 271–281, 1987.

[4] Bianco, L., Mingozzi, A., Ricciardelli, S. A Set Partitioning Approach to the Multiple Depot Vehicle Scheduling Problem. *Optimization Methods and Software* **3** 163–194, 1994.

[5] Blondin Massé, A., Chicoisne, G., Gargouri, Y., Harnad, S., Marcotte, O., Picard, O. How Is Meaning Grounded in Dictionary Definitions ? *Proceedings of the 3rd TextGraphs workshop on Graph-based Algorithms for Natural Language Processing (Coling 2008)*, Association for Computational Linguistics, 17–24, 2008.

[6] Bodin, L., Rosenfield, D., Kydes, A. UCOST : A Micro approach to a transit planning problem. *J. Urban Anal.* **5** 46–69, 1978.

[7] Bonami P. Étude et mise en oeuvre d'approches polyédriques pour la résolution de programmes en nombres entiers ou mixtes généraux. Thèse de Doctorat de l'Université de Paris 6, 2003.

[8] Bondy, A., Murty, U.S.R. Graph Theory. *Graduate Texts in Mathematics* **244** Springer, 2008.

[9] Caprara, A., Fischetti, M. $\{0, \frac{1}{2}\}-$Chvátal-Gomory Cuts. *Math. Programming* **74** 221–235, 1996.

[10] Caprara, A., Fischetti, M and Letchford A.N. On the Separation of Maximally Violated mod-k Cuts. *Math. Programming* **87** 37–56, 2000.

[11] Carpaneto, D., Dell'Amico, M., Fischetti, M., Toth, P. A Branch and Bound Algorithm for the Multiple Vehicle Scheduling Problem. *Networks* **19** 531–548, 1989.

[12] Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I. A Fixed-parameter Algorithm for the Directed Feedback Vertex Set Problem. *Proc. 40th STOC*, ACM Press, 177–186, 2008.

[13] Chu P.C, Beasley J.E. Constraint Handling in Genetic Algorithms : The Set Partitioning Problem. *Journal of Heuristics*, **11** 323–357, 1998.

[14] Chvátal, V. Edmonds Polytopes and a Hierarchy of Combinatorial Problems. *Discrete Math.*, **4** 305–337, 1973.

[15] Cormen T.H., Leiserson C.E, Rivest R.L., Stein C. Introduction à l'algorithmique. *DUNOD*, 2^e **édition**, 2002.

[16] Daily, D. P. The Extraction of a Minimum Set of Semantic Primitives from a Monolingual Dictionary is NP-Complete. *Computational Linguistics* **12** 306–307, 1986.

[17] Dantzig G.B., Wolfe P. Decomposition Principle for Linear Programs. *Operations Research* **8** No. 1 101–111, 1960.

[18] Dehne, F. K. H. A., Fellows, M. R., Langston, M. A., Rosamond, F. A., Stevens, K. An $\mathcal{O}(2^{\mathcal{O}(k)}n^3)$ FPT Algorithm for the Undirected Feedback Vertex Set Problem. *Theory Comput. Syst.* **41** 479–492, 2007.

[19] Desaulniers, G., Hickman, M. D. Public Transit. *Handbooks in OR & MS* **14** :Transportation (C. Barnhart, G. Laporte, eds.) 69–127. Elsevier, 2007.

[20] Desrosiers, J., Dumas, Y., Solomon, M., Soumis, F. Time Constrained Routing and Scheduling. *Handbooks in OR & MS* **8** :Network Routing (M. O. Ball, T. L. Magnanti, C. L. Monma, G. L. Nemhauser, eds.) 35–139. Elsevier, 1995.

[21] Dolan, E. E., Moré, J. J. Benchmarking optimization software with performance profiles *Math. Programming* **91** 201–213, 2002.

[22] Edmonds J. Maximum matching and a polyhedron with 0-1 vertices. *J. Res. Nat. Bus. Standards* **69B** 125–130, 1965.

[23] Elhallaoui I., Villeneuve D., Soumis F., Desaulniers G. Dynamic Aggregation of Set-Partitioning Constraints in Column Generation *Operations Research* **53** 632–645, 2005

[24] Even, G., Naor, J., Schieber, B., Sudan, M. Approximating Minimum Feedback Sets and Multicuts in Directed Graphs. *Algorithmica* **20** 151–174, 1998.

[25] Feillet D., Dejax P., Gendreau M., Gueguen C. An exact algorithm for the elementary shortest path problem with resource constraints : Application to some vehicle routing problems. *Networks* **44** 216–229, 2004

[26] Festa, P., Pardalos, P. M., Resende, M. G. C. Feedback Set Problems. *Handbook of Combinatorial Optimization (D.-Z. Du and P.M. Pardalos, Eds.)*, Kluwer Academic Publishers, Supplement vol. A, 209–259, 2000.

[27] Fischetti M., Dell'Amico M., Toth P. Heuristic algorithms for the multiple depot vehicle scheduling problem. *Management Science* **39** 115–125, 1993.

[28] Fischetti, M., Lodi, A., Martello, S., Toth, P. A Polyhedral Approach to Simplified Crew Scheduling and Vehicle Scheduling Problems. *Management Sci.* **47** 833–850, 2001.

[29] Fischetti M., Toth P. A new dominance procedure for combinatorial optimization problems. *Operations Research Letters* **7** 181–187, 1988.

[30] Forbes M.A., Holt J.N., Watts A.M. An exact algorithm for multiple depot bus scheduling. *European Journal of Operational Research* **72** 115–124, 1994

[31] Friedman E.J. Fundamental Domains for Integer Programs with Symmetries. *Springer-Verlag Berlin Heidelberg* **4616** 146–153, 2007

[32] Funke, M., Reinelt, G. A Polyhedral Approach to the Feedback Vertex Set Problem. *Proceedings of the 5th International IPCO Conference on Integer Programming and Combinatorial Optimization*, Springer-Verlag London, 445–459, 1996.

[33] Garey M.R, Johnson D.S. Computers and Intractability : A Guide to the Theory of NP-Completeness. *Freeman, Sans Francisco*, 1979.

[34] Garfinkel R.S., Nemhauser G.L. The Set-Partitioning Problem :Set Covering With Equality Constraints. *Operations Research* **17**, 848–856, 1969

[35] Ghoniem A., Sherali H.D. Defeating symmetry in combinatorial optimization via objective perturbation and hierarchical constraints. *IIE Transactions* **43**, 575–588, 2011.

[36] Gintner, V., Kliewer, N., Suhl, L. Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice. *OR Spectrum* **27** 507–523, 2005.

[37] Gomory R.E. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society* **64** 275–278, 1958

[38] Gomory R.E., Hu T.C. Multi-terminal network flows. *SIAM J. Applied Math.* **64** 551–570, 1961.

[39] Grötschel M., Jünger M., Reinelt G. A cutting plane algorithm for the linear ordering problem. *Operations Research* **32** 1195–1220, 1984

[40] Grötschel, M., Jünger, M., Reinelt, G. On the Acyclic Subgraph Polytope. *Mathematical Programming* **33** 28–42, 1985.

[41] Grötschel M., Lovász L., Schrijver A. Geometric Algorithms and Combinatorial Optimization. *Springer Verlag, Berlin-Heidelberg.*, 1988.

[42] Hadjar, A., Marcotte, O., Soumis, F. A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem. *Operations Research* **54** 130–149, 2006.

[43] Hoffman K.L., Padberg M. Solving Airline Crew Scheduling Problems by Branch-and-Cut. *Management Science* **39** 657–682, 1993.

[44] Irnich, S., Desaulniers, G., Desrosiers, J., Hadjar, A. Path-Reduced Costs for Eliminating Arcs in Routing and Scheduling. *INFORMS Journal on Computing* **22** 297–313, 2010.

[45] Kaibel V., Pfetsch M.E., Packing and Partitioning Orbitopes. *Math. Program.* **114** 1–36, 2008.

[46] Karp, R. M. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, Eds.)*, 85–103, 1972.

[47] Kliewer, N., Mellouli, T., Suhl, L. A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research* **175** 1616-1627, 2006.

[48] Land A.H., Doig A.G. An automatic method for solving discrete programming problems. *Econometrica* **28** 97–520, 1960

[49] Laurent, B., Hao, J.-K. Iterated Local Search for the Multiple Depot Vehicle Scheduling Problem. *Computers and Industrial Engineering* **57** 277–286, 2009.

[50] Lenstra J.K., Rinnooy Kan A.H.G. Complexity of Vehicle Routing and Scheduling Problems. *Networks* **11** 221–228, 1981.

[51] Levy, H., Low, D. W. A Contraction Algorithm for Finding Small Cycle Cutsets. *J. Algorithms* **9** 470–493, 1988.

[52] Lewis M., Kochenberger G., Alidaee B. A new modeling and solution approach for the set-partitioning problem. *Computers and Operations Research* **35** 807–813, 2008

[53] Lin, H.-M., Jou, J.-Y. On Computing the Minimum Feedback Vertex Set of a Directed Graph by Contraction Operations. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* **19** 295–307, 2000.

[54] Löbel, A. Optimal Vehicle Scheduling in Public Transit. Ph.D. thesis, Technische Universität Berlin, Berlin, Germany, 1997.

[55] Löbel, A. Vehicle Scheduling in Public Transit and Lagrangean Pricing. *Management Sci.* **44** 1637–1649, 1998.

[56] Méndez-Dìaz I., Zabala P., A Branch-and-Cut Algorithm for Graph Coloring. *Discrete Appl. Math.* **154** 826–847, 2006.

[57] Nemhauser, G. L., Wolsey, L. A. *Integer and Combinatorial Optimization*. Wiley, New York, 1988.

[58] Nemhauser, G. L., Sigismondi, G. A Strong Cutting Plane/Branch-and-Bound Algorithm for Node Packing. *J. Oper. Res. Soc.* **43** 443–457, 1992.

[59] Orenstein, T., Kohavi, Z., Pomeranz, I. An Optimal Algorithm for Cycle Breaking in Directed Graphs. *Journal of Electronic Testing : Theory and Applications* **7** 71–81, 1995.

[60] Ostrowski J., Linderoth J., Rossi F., Smriglio S. Orbital Branching. *Proc. 12th Conference on Integer Programming and Combinatorial Optimization (IPCO). LNCS, Springer, Heidelberg*, 2007.

[61] Oukil, A., Ben Amor, H., Desrosiers, J., El Gueddari, H. Stabilized Column Generation for Highly Degenerate Multiple-Depot Vehicle Scheduling Problems. *Computers & OR* **34** 817–834, 2007.

[62] Padberg, M. W. On The Facial Structure of Set Packing Polyhedra. *Math. Programming* **5** 199–215, 1973.

[63] Padberg, M. W., Rao, M. R. Odd Minimum Cut-sets and b-matchings. *Mathematics of Operations Research* **7** 67–80, 1982.

[64] Pepin, A.-S., Desaulniers, G., Hertz, A., Huisman, D. Comparison of Heuristic Approaches for the Multiple Depot Vehicle Scheduling Problem. *Journal of Scheduling* **12** 17–30, 2009.

[65] Pulleyblank, W. R., Edmonds, J. Facets of 1-Matching Polyhedra. *Hypergraph Seminar* (C. Berge and D. Ray-Chaudhuri, eds.), 214–242. Springer, Berlin, Germany, 1974.

[66] Rebennack, S. Maximum Stable Set Problem : A Branch & Cut Solver. Diploma Thesis, Ruprecht-Karls-Universität Heidelberg, Heidelberg, Germany, 2006.

[67] Ribeiro, C., Soumis, F. A column generation approach to the multiple depot vehicle scheduling problem. *Operations Research* **42** 41–52, 1994.

[68] Sherali H.D., Smith J.C. Improving Discrete Model Representations via Symmetry Considerations. *Management Science* **47** 1396–1407, 2001.

[69] Thompson G. L. An Integral Simplex Algorithm for Solving Combinatorial Optimization Problems. *Computational Optimization and Applications* **22** 351–367, 2002.

[70] Walker W.E. A method for obtaining the optimal dual solution to a linear program using the Dantzig-Wolfe decomposition. *Operations Research* **17** 368–370, 1969.